Task Manager

Kadar Tibor Manasse
Technical University of Cluj-Napoca
Group 30433

Overview

The project aims to implement a similar tool to the Windows Task Manager. In essence the tool offers functionalities which enable the user to see the CPU load, processes running and offers possibilities to manage them. The tool shows a list of currently running processes along with a percentage which represents how much does the process use the CPU. The Windows Task Manager offers additional features like like displaying the percentage used from other hardware resources such as Memory, Disk, GPU and so on.

This project will focus mainly on implementing the features that have a direct interaction with the CPU that is: monitor the processes and their CPU usage in percentage, display a performance graph over time of the overall CPU usage. Based on the difficulty of the implementation new features may be added.

Goals

The tool should have the following functionalities:

1. The main objective of the tool is to offer a view on what does the CPU do and how hard is it working.
2. Monitor the processes and display a percentage representing the CPU usage for each process, as well as an overall CPU usage.
3. Show a Performance tab which displays a graph of the evolution of the CPU usage based on the percentage over a minute.

Specifications

Design and implement basic functionalities regarding processes running on the CPU. The functionalities include:

- Display the percentage of CPU usage by each process
- Display the CPU usage evolution on a period of one minute

- Add other functionalities from the Task Manager, such as the possibility to kill a process

It will be developed using the built in calls in the operating system using C++.

Bibliographic Study

How others do it:

Useful information about implementing the tool can be found at the following link:

MSDN documentation on Processes and Thread functions

Stack Overflow conceptual solution

As this documents describe there are functions in the operating system which help achieving this task:

Useful functions in the PSAPI:

1. **EnumProcesses**: Retrieves the process identifier for each process object in the system. The identifiers are the process handles, don't forget to close them.
2. **GetPerformanceInfo**: Retrieves the performance values contained in the **PERFORMANCE_INFORMATION** structure.
3. **PERFORMANCE_INFORMATION** contains information about the total number of processes and threads running on the CPU.
4. **GetProcessTimes**: retrieves a structure from which we can extract the wall time of the process, time that it has run in kernel mode, and time that it has run in user mode.
5. **GetSystemInfo:** Used to get the number of logical cores.

There are two ways to measure time: CPU time - the time actually spent by CPU executing method code. Wall time - the real-world time elapsed between method entry and method exit

To calculate the cpu usage in percentage we should do the following:

CPU usage = (CPU time) / (# of cores) / (wall time)

Furthermore after consulting the MSDN page on **Performance Counter Helper** Interface. This interface provides a frame for querying the CPU counters.

1. Create a query
2. Add counters to the query
3. Collect the performance data
4. Display the performance data
5. Close the query

For this operations there are defined functions in the PCH library.

This is used in the Task Manager and thus it is recommended by the MSDN community.

As i have found following the above documentation the task is doable in the following manner. There are functions which can help in the Win32 api to obtain the required raw information which can then be processed to obtain the results in the way we want. There are several ways and libraries that help get the necessary information but the recommended one is the PCH library.

Requirements

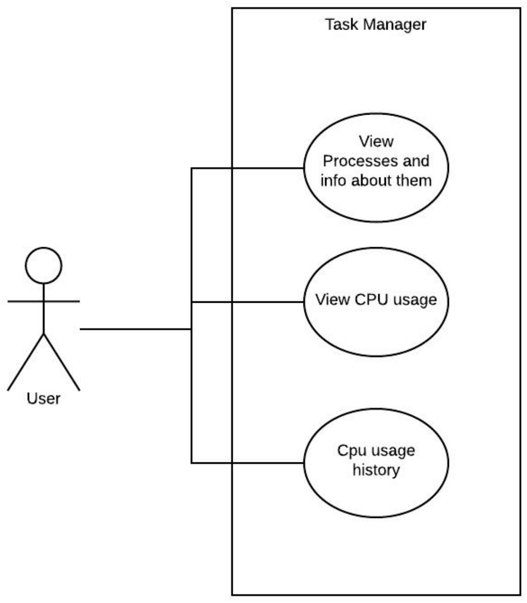
Design and implement basic functionalities regarding processes running on the CPU. The functionalities include:

Display the percentage of CPU usage, Ram usage and Virtual memory usage by each process

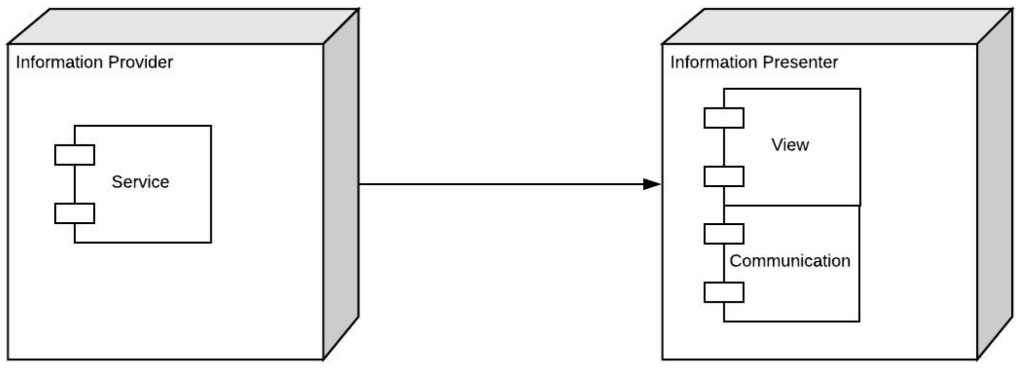
Display the CPU usage evolution on a period of one minute.

It will be developed using the built in calls in the operating system using C++.

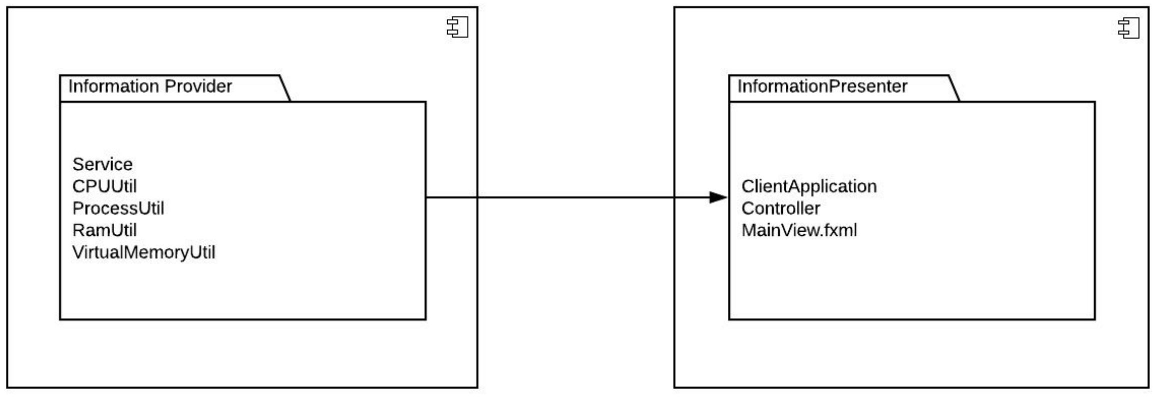
Use Case Model



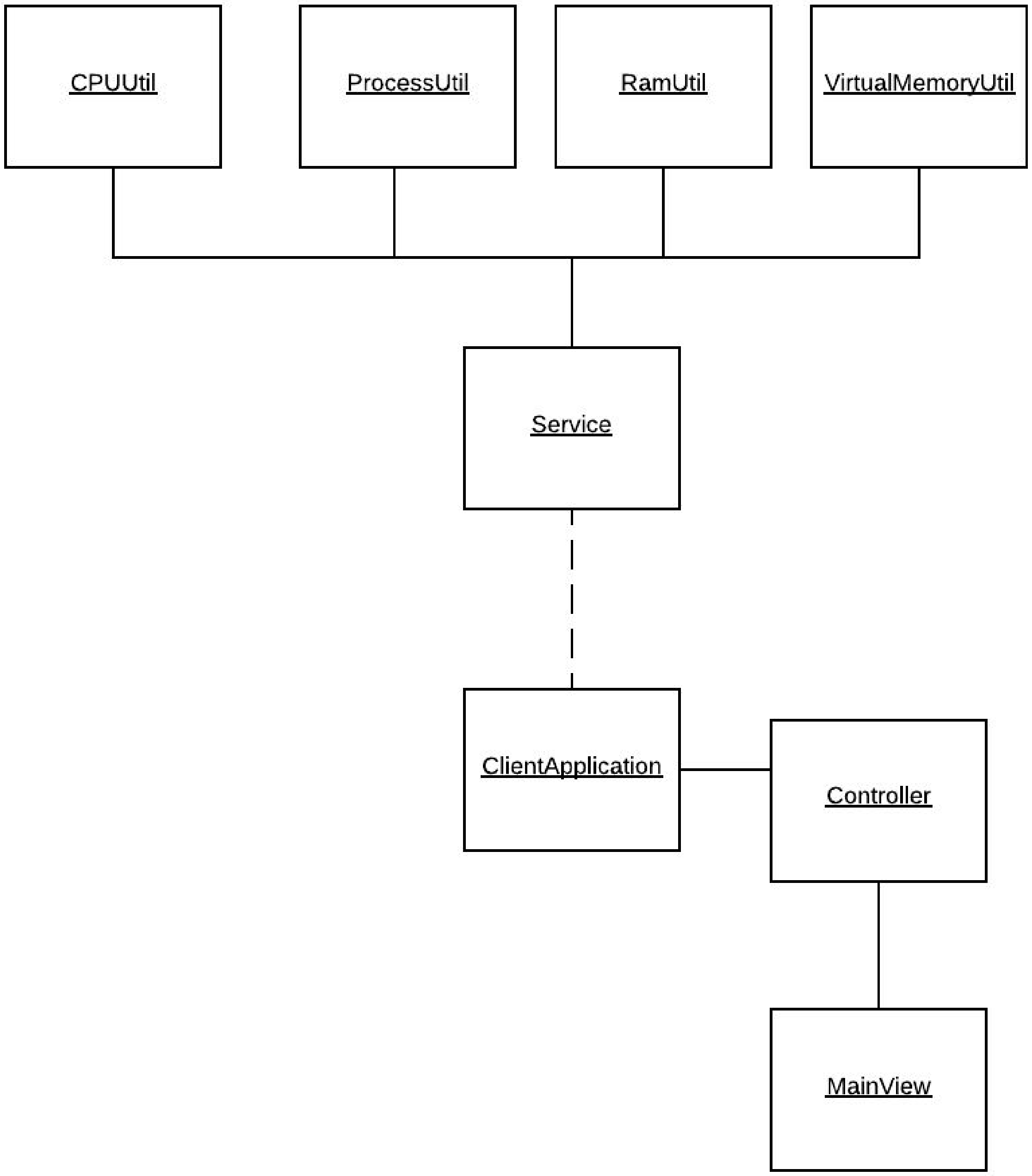
Conceptual Architecture



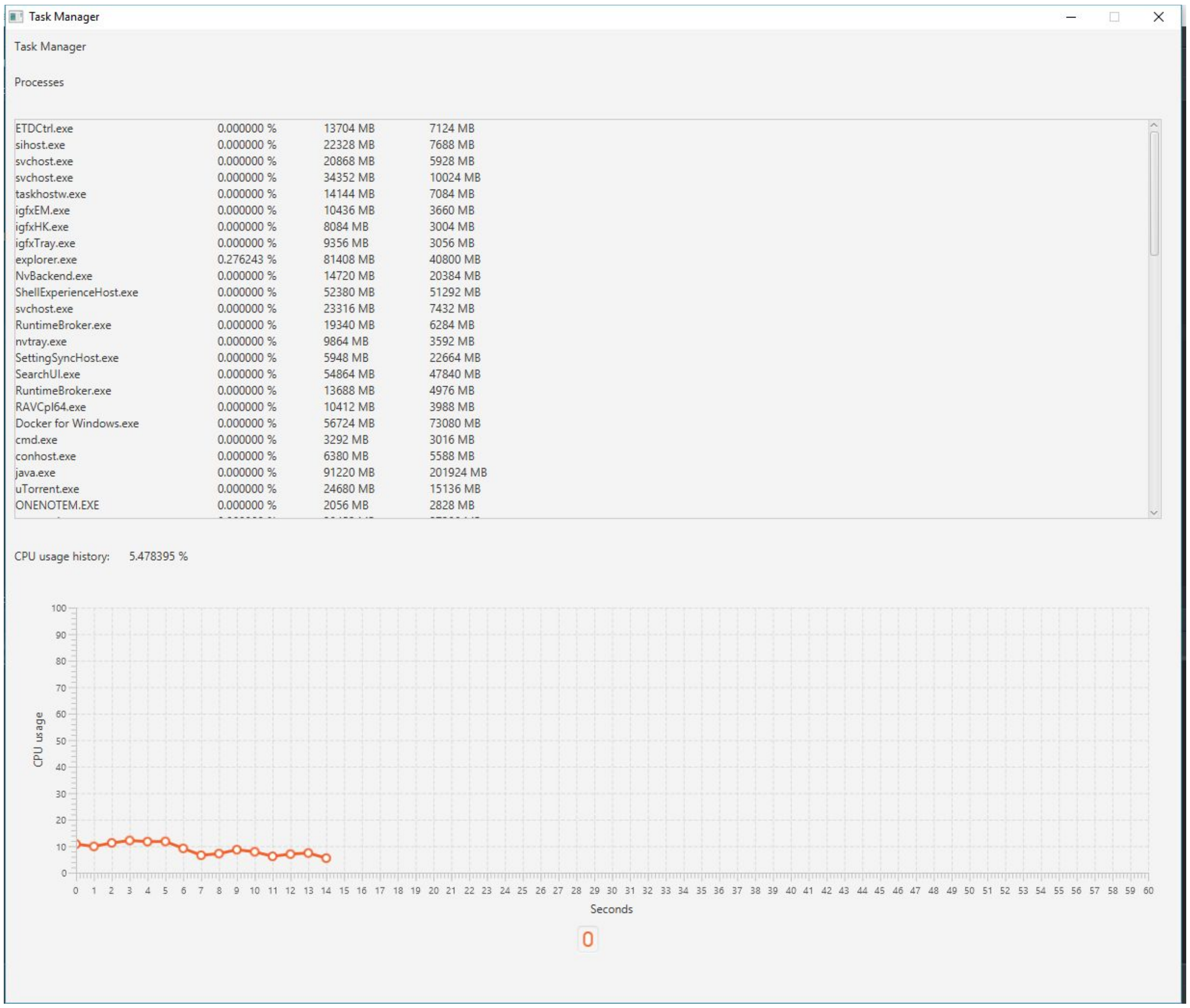
Package Diagram



Class design



GUI



ETDCtrl.exe	0.000000 %	13704 MB	7124 MB
sihost.exe	0.000000 %	22328 MB	7688 MB
svchost.exe	0.000000 %	20868 MB	5928 MB
svchost.exe	0.000000 %	34352 MB	10024 MB
taskhostw.exe	0.000000 %	14144 MB	7084 MB
igfxEM.exe	0.000000 %	10436 MB	3660 MB
igfxHK.exe	0.000000 %	8084 MB	3004 MB
igfxTray.exe	0.000000 %	9356 MB	3056 MB
explorer.exe	0.276243 %	81408 MB	40800 MB
NvBackend.exe	0.000000 %	14720 MB	20384 MB
ShellExperienceHost.exe	0.000000 %	52380 MB	51292 MB
svchost.exe	0.000000 %	23316 MB	7432 MB
RuntimeBroker.exe	0.000000 %	19340 MB	6284 MB
nvtray.exe	0.000000 %	9864 MB	3592 MB
SettingSyncHost.exe	0.000000 %	5948 MB	22664 MB
SearchUI.exe	0.000000 %	54864 MB	47840 MB
RuntimeBroker.exe	0.000000 %	13688 MB	4976 MB
RAVCpl64.exe	0.000000 %	10412 MB	3988 MB
Docker for Windows.exe	0.000000 %	56724 MB	73080 MB
cmd.exe	0.000000 %	3292 MB	3016 MB
conhost.exe	0.000000 %	6380 MB	5588 MB
java.exe	0.000000 %	91220 MB	201924 MB
uTorrent.exe	0.000000 %	24680 MB	15136 MB
ONENOTEM.EXE	0.000000 %	2056 MB	2828 MB

CPU usage history: 5.478395 %

Implementation Details

The project proved to be more difficult than expected in the way that presenting the informations in a clear and concise way, that is making the GUI in C++, was challenging, hence the decision was made to split the application in two parts.

The Information Provider

As seen in the conceptual architecture the project has 2 main modules, the information provider module written in C++ contains the service classes which implement the mechanisms by which the informations regarding the CPU, RAM, Virtual Memory are retrieved.

This module has 4 important components which are separate classes:

CPUUtil - this class has the responsibility to retrieve the cpu usage of a certain process, the two functions that it provides are **getTotalCPUUsage** and **getCPUUsageForProcesses**. The two functions work in the following way:

Retrieve the cpu times: Kernel User and Idle

Wait for an interval of time - i used one second

Retrieve again the cpu times: Kernel User and Idle

Calculate how much time the processor was working on that process or in general in this interval and convert it into a percentage.

RamUtil and VirtualMemoryUtil - this classes implement functionalities that retrieve all the physical or virtual memory used by the system and also the memory used by one process. The functions work the following way:

This service class makes use of the **GlobalMemoryStatusEx** function and the **MemoryStatusEx** structure which provides useful information regarding the memory in the system.

ProcessUtil - this class implements functionalities regarding the processes, firstly a great function of this class is to retrieve the number of processes in the system and more over determines which of them can be viewed and operated on, that is for which processes do I as a user have access.

This module furthermore has to tie this functionalities together and that is the responsibility of the Service class. The workflow of this module is the following, retrieve the processes that can be analyzed, calculate the CPU usage for each of them and then loop through the processes to find all the relevant information.

So far so good, but as it usually happens a problem emerged, how should i present this information? Upon 3 weeks of working on strategies a solution emerged and that was communicate with the Information Presenter module via **sockets**. Hence the Service class obtained more responsibilities that involve creating a **ServerSocket**, accepting a connection and sending to the connected **client socket** the relevant information.

The Information Presenter

This module has the responsibility to present the information in a way that the user can understand it. The decision was made to implement this module using JavaFx. While all this is simple the complicated part was to implement the Threading mechanism that will communicate with the server socket in a way to not block the UI. So how does it work:

As for every JavaFx application there is a UI thread which will do work regarding the UI, updates controls and such. This thread should not be blocked because that would yield a frozen and not usable UI. So for this reason the communicating mechanism is implemented in a separate non blocking thread. This thread will update the data which is tied to the UI. This data is made observable meaning that whenever it changes the UI will update accordingly.

This module represents the data in the following way:

In a scrollable region the processes are enumerated along with the ram and virtual memory usages.

Under this region a Line Graph is displaying the evolution of the CPU usage in a span of a minute. The graph advances from left to right and when it reaches the 60 second interval it starts shifting to the left in order to keep in the field of view the relevant last 60 seconds.

Testing Details

The project proved to be error prone in lots of ways. By testing the application it gotten ever so slightly better but there were quite the bugs.

Testing the Information Provider

This unit was under test the most, since it is the core of the project, and the information should be as correct as possible.

Taking the Task Manager as the standard was both a mistake but also helped improve the understanding of the problem. Because this unit implements the functionalities regarding the processes in a different way than the Task Manager it is expected to have different values, but from where do this different values come from? And by analysis of the correctness of the algorithms i optimized it and got almost the same results, CPU Usage is accurate with a possible + - 1% due to it being not able to get information about processes which run in other privilege modes.

Another problem encountered was the RAM usage being off, only to find out later that it is indeed correct but the Task Manager uses both RAM and Virtual Memory in the RAM section combined.

Testing the information presenter was a challenge also, since the communicating mechanism being on another thread there were lots of thread communication bugs, for example updating 2 times the observable data and thus updating twice the graphic.

Bad implementation of the CPU usage evolution graph, had to change to a line graph because it wasn’t very understandable and intuitive.

Testing the whole system together both the Information Provider and Presenter proved to be the most challenging because it involved lots of restarts.

Bugs that emerged from the 2 applications working together were related to them being out of sync and thus reading a number from a string would crash the Presenter.

Weird bug regarding process that dies right when we measure its performance. This is not fixed yet. This is a vulnerable point of the application, but does not happen often.

Conclusion

As a conclusion i would like to state my impressions on this project. I liked the idea of working on this project and it was always something that i wondered how it was made. I expected to be more exact but as it turns out it is just as it is, different implementations work in different ways. C++ GUI was a bad experience but in the end it got better. Overall the project offered a lot to learn and for that i’m grateful.