**Computer Science: Operating Systems**

**Final Report on ‘Process Status’ for**

**Windows command line/PowerShell**

**in Visual Studio, C++**

Tim Doan

Tim Doan, Computer Science 4270, Class 001

481 Hunters Mill Cove

Tim Doan,

[tdoan@memphis.edu](mailto:tdoan@memphis.edu)

**Abstract.** Using numerous function calls of the C++ library, I am able to call them using Visual Studio 2017 in order to write a few methods. I have had no experience with C++ prior to this project. The difficulty was learning C++ from scratch as well as learning the function calls needed to run everything. These methods are mainly the ‘PrintProcessAndID’ method as well as the ‘Main Method.’ The ‘PrintProcessAndID’ method obtains a list of processes currently running. Next is obtaining the handle by calling Open Process, which is a core function call. Using this handle, the method is able to get the process name by calling from the ‘GetModuleBaseName’ function. If the method should fail in getting a process name because of user-level restrictions for the process, it will print out “unknown” for the name. The ‘PrintProcessAndID’ method will be run in the main method and display both the process name and process ID for the list of processes.

**Keywords:** Function Call - A call that passes control to a subroutine; after the subroutine is executed control returns to the next instruction in main program.

Method – Procedure that in defined in a class that is often associated with an object.

Handle – An abstract reference to a resource.

Process – An instance of a computer program that is being executed.

**1 Introduction**

The process status command, otherwise known as the ‘ps’ command, in a Linux/Unix operating systems will display all of the active processes currently running on the system and any relating information associated with them. On windows, the task manager will display some of the running processes on your local computer, however there are types of Malware out there that can hide itself from task manager by manipulating the file system structure. These types of Malware are called rootkits and are usually embedded deeply in the operating system.

**1.1 Factors to the Solution**

By using the process status command or a similarly designed one for the windows equivalent, one may be able to bring up all the currently running processes for analysis. Once these active processes are identified, information about the process is also given and one may be able to use that information in order to reverse engineer the rootkit or even understand its behavior more clearly. This is a great way to understand how hackers or crackers design viruses or rootkits and stop them from happening in the future.

**1.2 The Solution**

In the case of this project, I was tasked with solving this problem of being able to look at processes on a deeper level for Windows PowerShell. In other words, I was to create a program that would act and function very similarly to how the task manager runs for Windows.

My code captures a list of all the running processes at the time it is run successfully and prints them to console. This will show any process (and the name) that the user has access to as well as the process ID.

**1.3 Structure of the Paper**

For the remainder of my paper, I will discuss my related works and experience with my code as well as C++. I will also heavily go over the design of the code I have presented as well as provide visual materials for comparison of its effectiveness against other forms of the process status command. Following up, I will proceed to discuss the evaluation and analysis of my design and see how it compares. I will evaluate whether or not it effectively does its job and how useful it is as a solution in solving the problem mentioned initially.

As a bonus, I have included a ‘Future Works’ and ‘Conclusion’ section in which I will talk about the features I will implement for my program later on. This will be additional content and fixes that I will want to apply. As for the conclusion, I will summarize an analysis on what I will address for the paper.

**2 Related Work**

Currently, I am a senior level undergraduate at the University of Memphis. I have been in the software engineering course so I have prior exposure to web development as well as github. I have fiddled with C before on personal projects but have never dealt with creating something that deals directly with the OS in C++.

Prior to this project, I have had very little experience with C++ and thought learning it would be extremely similar to C. In teaching myself C++, I had to do a lot of my research online and from scratch. Learning how to code in C++ was not as difficult as I had initially thought since I was somewhat familiar with it already, however the main problem was with the function calls for C++.

C++ has many function calls and reading up on all of them was what I had to focus on for the majority of my project. In order to learn how to make the function calls with the different C++ headers, I had to grasp a general idea of what headers I initially needed. It was frustrating to learn that some of the headers I thought I would need at first did not end up pertaining to the final code in some cases. These headers provided for function calls that I could use, and they helped me significantly in coding my project.

**2.1 Additional Projects**

For the personal projects I have created, I have programmed in C and C++ briefly to design code for a Magic Mirror. This is essentially a computer screen behind two-way glass that makes it seem like a smart mirror. The coding I have done with the mirror is specifically in the helpers and modules. The Magic Mirror runs using a full screen interface and the modules I programmed directly affect what is displayed on the screen. The helpers I coded simply call on other functions with unique ID numbers that can additionally display more features if the user wants to.

**3 Design**

My program was done in Visual Studio 2017. This is a programming environment in which the user is able to compile and run code written in C++. It contains a ‘.cpp’ file as the main portion where you can program as well as the ‘.h’ file where you can place any necessary headers.

For my program, I created two methods that utilizes the various function calls inside of the C++ libraries. The first method was ‘PrintProcessAndID’ which does as the name suggests, and prints the processes’ names and ID to the console. This method goes about doing this by getting an input (the process ID) and passing it through. This happens on every process that is running, and the code follows up by getting a handle on the process it is looking at using ‘OpenProcess.’ Next, ‘EnumProcessModules’ is called along with ‘GetModuleBaseName’ in order to get the process’s name. Using the ‘tchar’ header, the program is able to write out the process’s name. The handle is then closed, and this step repeats for every other process that is going to be called in the operating system.

The main method creates a ‘for’ loop with the number of process IDs that are running and iterates through them to find out how many there are. The program then returns this list by calling ‘PrintProcessAndID’ on all of them to print to the console.

**3.1 Headers**

I included the following headers because they were deemed relevant to my program:

#include <windows.h>

#include <stdio.h>

#include <tchar.h>

#include <psapi.h>

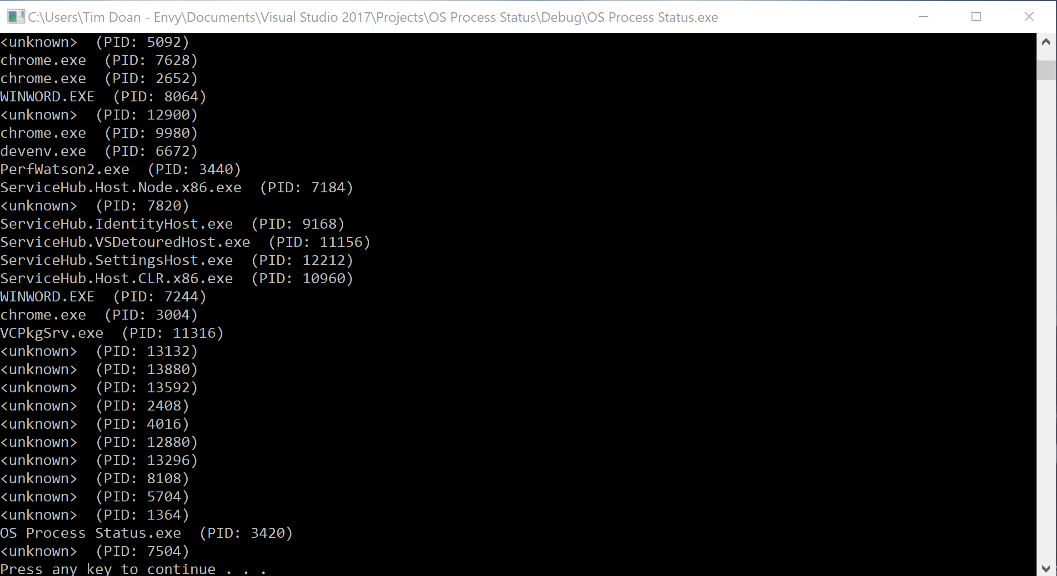
#include <tlhelp32.h>

The ‘windows’ header was necessary because I was running this on windows and C++ needs access to all of the Windows API declarations. The ‘stdio’ header was necessary because it allows me to print to the console, or put any output at all. Things included in the ‘stdio’ (standard input/output) header would allow me to run commands of code like ‘printf’ or ‘cout’ which allow me to display and print to the console. Furthermore, I included the ‘tchar’ header because it allows me to write, compile, and maintain a single source code file to run with routines that are specific to different character sets. Without this header, I would not be able to read in specific characters at all.

Additionally, I included the ‘psapi’ header because this header allows me to access a lot of information pertaining to processes and other things in the API functions. This is probably the most vital of my headers because it allows for the main “guts” of my code to compile and run. Similarly, the ‘tlhelp32’ header works in tandem with the ‘psapi’ header and gives me a lot of information pertaining to processes. This allows for taking snapshots of the processes in a list and helped me a lot during my coding.

**3.2 Implementation**

Pictured below is an image of my program and its output. This will display all of the processes that the user has access to and print the name and ID to the console at hand. In this case, it is using Visual Studio 2017.



**Fig 1.** This figure shows the complete display of all the processes currently running on my operating system in numerical order (ascension by process ID) as well as the name. The reason for the ‘unknown’ names are explained in the following section.

**3.3 Reasoning**

The reason for the console displaying ‘unknown’ for some process names is because of the following lines of code:

TCHAR ProcessName[MAX\_PATH] = TEXT("<unknown>");

Some processes are at a level some users will not be able to have access to. Because of this restriction, I did not want the method to error out on obtaining the process name and return unreadable characters. Initially, I did not have this line of code in the program, and whenever the program ran, it would display a string of long, and mostly unreadable gibberish (did not seem like English/readable to me) for the process name. I defaulted to this because the process was beyond the user accessibility and would print an ‘unknown’ process instead.

Furthermore, I also included the following line of code because it was needed for user to see the console:

system("pause");

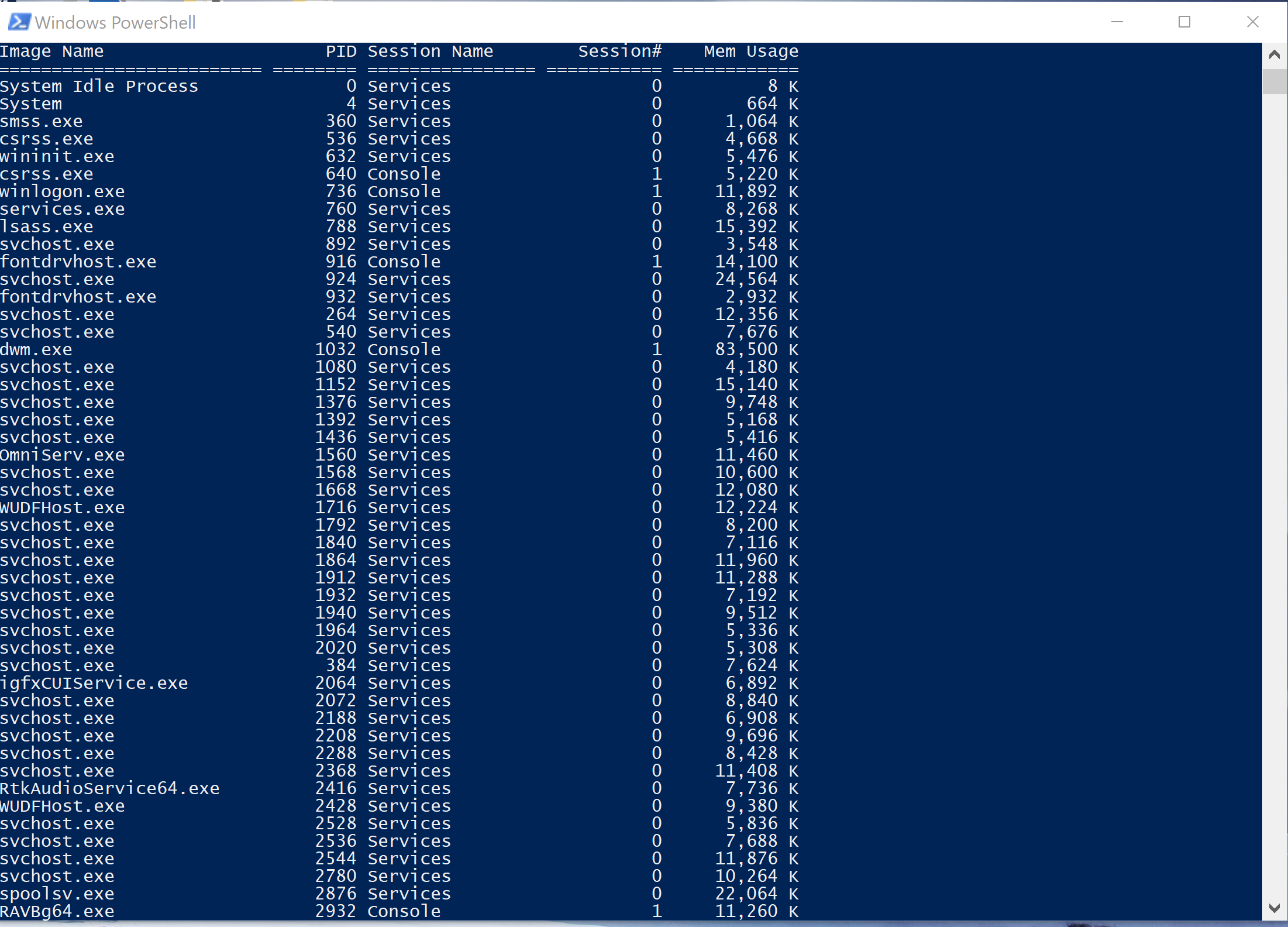
This line specifically causes the console to ‘pause’ and not close out. The program does its job of printing every process and process ID to the console, however, it closes out once it is finished. For viewing purposes and simplicity of the user, I decided to add this so that the user may be able to view each process in detail before deciding when to close the program/code.

**4 Evaluation/Analysis/Results**

As for evaluating my code, I would say I have successfully created a process status for a Windows equivalent. Using C++ has definitely expedited my work and provided substantial leaps in coding. I am glad that this was done in C++ because if it were done in another language, I’m afraid I would not have the resources to be able to write such compact amount of code and successfully create a process status version for windows even if I had experience with it. The function calls built into the C++ libraries were extraordinarily helpful.

**4.1 Comparison**

Below is a screenshot of the output of Window’s version of process status, which is ‘tasklist.’ I will use this as a comparison for my program.



**Fig. 2** This figure shows the complete list of processes currently running on my operating system using the ‘tasklist’ command in Windows PowerShell. This command is the Windows equivalent of ‘process status’ on a unix/linux console.

When looking at the display using the ‘tasklist’ command, I can clearly see that every process is printed with their names and ID. In addition to this, I can also identify the session name, the session number, and the memory usage of that process as well. This is useful information that my running version does not contain. My program simply gets the process name and ID.

Since I was using Visual Studio 2017, it restricted my program’s access to all the process names, however, if I had kept researching the method calls and header functionality, I might have been able to make the program have admin level access to the operating system in order to get a “complete” list.

Another issue my program does not address is exactly a fault in the programming. The initial problem was to identify a rootkit by its name or functionality and see whether or not it is causing harm to my operating system, or some other ill intent. However, since my program can’t print some names to the console, then some rootkits/malware will be able to get through. In the case the malware is at a higher level of access than my program, then I will not be able to identify it, and its process name will simply be “unknown” to me.

**5 Future Works**

In the future, I hope to work on this code more to improve it. I have identified the main problems in the analysis section, and I would like to address that here as well. For future reference, I will start by checking to see if I can recreate this program or run this program at an administrator level. This will improve what the processes the program has access to. This will be my priority in terms of what I am working on in the future.

**5.1 Additional Features**

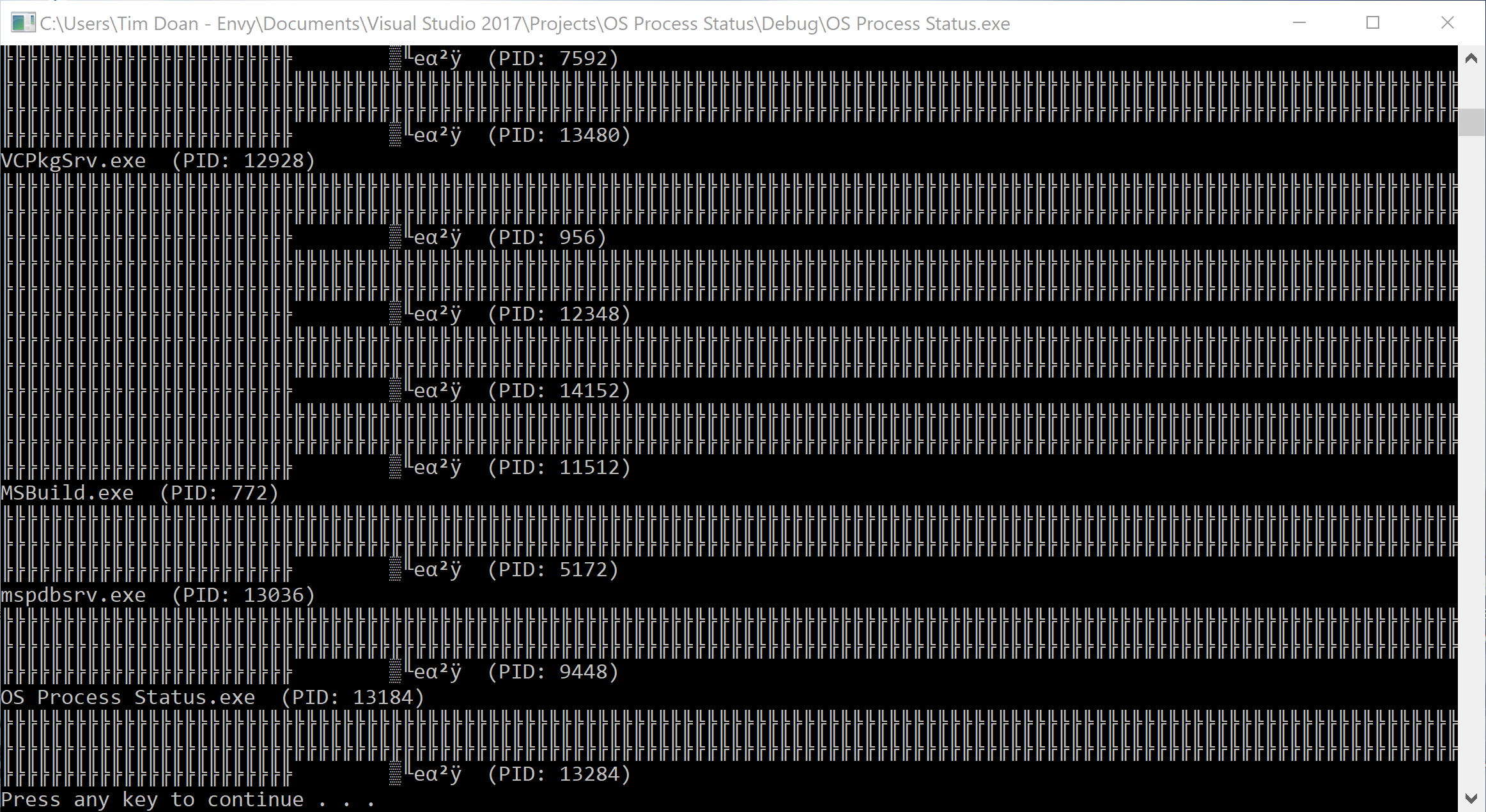
For more features that I would like to implement into my program, I would like for the program to eventually be able to display the session number and session ID as well. This was something I was looking in prior to submitting the project, however I never got around to it because I did not understand the libraries fully at the time. If I were to continue in this vein, I would assume it has to do with the headers I had previously included. This would allow for me to call on the process session and session ID because the header already encompasses a lot of these process information segments. This might have to deal with enumerating processes or information inside of the handle obtained from the process.

**6 Conclusion**

The code I have submitted contains a very basic process status version for Windows. It gives the basics of just the process name and process ID. It does not completely address the initial problem posed at the beginning of this paper, however it does address some parts of it and enables the user to access their level of restriction based processes. Later on, I could potentially work on this more to give it more access that is less restricted in order to solve the problem of rootkits on this particular laptop.

**7 Appendices**

Pictured below is the output display without the ‘unknown’ line to the code:



**Fig. 3** Above is a figure displaying the console output without the ‘unknown’ text line. This causes errors in reading process names because of user level access restrictions.