Iraqi Journal of Science, 2022, Vol. 63, No. 11, pp: 5046-5057
DOI: 10.24996/ijs.2022.63.11.39

ISSN: 0067-2904

Root Cause Analysis And Improvement In Windows System Based On Windows Performance Toolkit WPT

Murtadha J. Assi*, Assmaa A. Fahad, Basad Al-Sarray
Department of Computer science, University of Baghdad, Baghdad, Iraq

Received: 15/9/2021 Accepted: 7/4/2022 Published: 30/11/2022

Abstract

Performance issues could be appearing from anywhere in a computer system, finding the root cause of those issues is a troublesome issue due to the complexity of the modern systems and applications. Microsoft builds multiple mechanisms to make their engineers understand what is happening inside All Windows versions including Windows 10 Home and the behavior of any application working on it whether Microsoft services or even third-party applications, one of those mechanisms is the Event Tracing for Windows (**ETW**) which is the core of logging and tracing in Windows operating system to trace the internal events of the system and its applications. This study goes deep into internal process activities to investigate root cause analysis on Windows 10 Home 20H2, core i5 processor with 4 cores and 8GB of RAM. After simulating workload to get a performance issue, that makes the system and application get unresponsive, then using Windows Performance Toolkit **WPT** to trace and analyze the event log for root cause investigation. Our results demonstrate analysis works using WPT for decision making such as the reasons of underutilization on CPU and disk, labeling the highlighted patterns, the unbalanced use of system calls in memory, and deciding that the usage preview is the best way to get an idea about applications behavior inside Windows systems resources. Overall improving resources utilization usage and identifying the cause of slowing memory allocation, inefficient disk usage, and throughput.

Keywords: ETW, WPT, WPA, WPR, trace, provider.

تحليل اداء نظام وندوز وتحسينه باستخدام أدوات تحليل النظام

مرتضى جبار عاصي*, أسماء عبد الله فهد , بسعاد السراي
قسم علوم الحاسوب, كلية العلوم, جامعة بغداد, بغداد, العراق

الخلاصة

يمكن أن تظهر مشكلات الأداء من أي مكان من محتويات الأنظمة ، ويعد العثور على السبب الجذري لهذه المشكلات مشكلة مزعجة نظرًا لتعقيد الأنظمة والتطبيقات الحديثة. قامت شركة مايكروسوفت ببناء آليات متعددة لجعل مهندسيها يفهمون ما يحدث داخل أنظمة وندوز وتطبيقاتها ، وإحدى تلك الآليات هي تتبع الأحداث لنظام وندوز والتي تعد جوهر التسجيل والتتبع في نظام التشغيل وندوز لتتبع الأحداث الداخلية للنظام وتطبيقاته. تتعمق هذه الدراسة في أنشطة العملية الداخلية للتحقيق في تحليل السبب الجذري لمشاكل الاداء في

*Email: murtadha.jabbar.aasi.karam@gmail.com

نظام التشغيل وندوز 10 بعد محاكاة توليد عبء العمل للحصول على مشكلة في الأداء ، مما يجعل النظام والتطبيق لا يستجيبان ، ثم استخدام مجموعة أدوات تحليل نظام وندوز لتتبع وتحليل سجل الأحداث للسبب الجذري. توضح نتائجنا مهارات التحليل لاتخاذ القرار بشأن أنشطة النظام والتطبيق وتأثيرها على أعراض مشكلات الأداء ، لتحسين استخدام استخدام الموارد وتحديد سبب إبطاء تخصيص الذاكرة ، واستخدام القرص غير الفعال ، والإنتاجية.

Introduction

System performance studies the performance of an entire computer system, including all software and hardware contents from the storage device into the application software because it could impact the performance[1]. In software engineering terms poor performance can affect each of product usability, related customer productivity, wall clock time restrictions (functionality of applications from the start till the completion).

In the operating systems world performance have multiple areas and concepts like it deal with resource efficiency and its utilization, how fast tasks are complete, how the system and its application responsive for the end-user, and how smooth user experiences are. In software engineering terms the poor performance affects each of product usability, related user productivity (the monetary value of the time), wall clock time constraints (functionality)

The growing size and complexity of the recent software systems make the development a tough challenge. Systems and the performance of their applications are in challenging due to many reasons including subjectivity, complexity, could not have a single root cause, and it is frequently associated with multiple reasons. The application today becomes more complex due to developing over time, which causes difficulties for the performance analyst to understand the underlying events. So, the analyst needs to capture low-level system activities to find clues of performance issues and the factors responsible for unexpected behavior.
The ability to trace programs and monitor resource usage becomes an important feature in the management and development of recent reliable applications. The current big applications become more complex and hard to manage, analyze, troubleshoot, track internal activities or predict their execution state and accuracy [2].

The complexity in the recent applications has been increased nowadays, which makes their operations cause more challenges on system reliability and performance needs. The complexity in software operations causes increasing anomalies inside system operations [3]. The ability to log application activities and track its metrics becomes important in the recent development application, capturing mechanisms in modern systems, to provide rich log files that contain important system contents such as system performance counters, events tracing, process activities. Logging represents the fundamental source of wealth information on the activities of systems that mostly be helpful for its events, resource usage, software, and system degradation. Logging records runtime information which is helpful for support engineers, administrators, and developers to analyze their systems to highlight its behaviors and root cause analysis [4]. Multiple papers on the analysis of Performance representing problem-align and highlight issues like Software Aging issues [5],[6], detecting malware attacks on the systems [7], overview on multiple processes on windows event logging environment [8], etc.

Tom P. in Intel used windows tools for resources utilization to see if certain resources like CPU Usage whether 100% utilized or above the threshold values, WPT has been inducted to record and analyze the related metrics that impact the performance aspect [9].

In Windows systems, there are two platforms for measuring performance are Performance counter for Windows (**PCW**) and Event tracing for Windows (**ETW**). **PCW** is a great platform for monitoring Windows performance, it gives the analyst a sample as taking a snapshot of performance measurement over time (long period e.g., min. hrs., days), it could monitor any system counters. **ETW** is a general-purpose platform for gathering diagnostic information especially low-level system and application activities, ETW also could be used for high frequently samples (e.g., sec, min, hrs. like to diagnose what CPU is doing every millisecond, which process consumes Memory, and for what reason, it could help the analyst to not back to source code by giving low-level information in the call stack)
The following reminder of the paper is as follows, In **Sections I**, Fundamental of Windows Internals. Event tracing for Windows in **Section II**, **Section III** represent the Methodology analysis. Lastly, **Section IV** the Results

I. Fundamentals to Windows Internals and Performance

The existence of any Performance issue in a modern big system like Windows OS could have a practical reason, there should be root causes that make them appear on the user interface UI. To analyze the performance issues on Windows OS, then it is important to get an overview of Windows system fundamentals and forward to deeper level knowledge like how data is coming from, and how it works under the hood to understand the performance issue, especially when dealing with windows performance analysis.

A. Fundamental to Windows Internals

It is hard to explain the whole deep windows internals because it is pretty heavy but to better understanding Windows performance firstly take a look to Internal design and components of its Architecture from high-level view, on Windows the kernel does not provide a documented interface to the applications developers, instead Windows provide user space libraries which have .DLL (Dynamic Link Libraries) extensions, and these libraries provide various functions and calls which are represent the interface between the application and the kernel, high level structure view of windows architecture that there are application is services that calls functions provided by set of DLLs which could be C++ and other libraries that the apps depend on, and those libraries calls the underlying functions of Win32 API which is the main application programming interface that Microsoft documents exposes, and it is provided by standard DLLs which include kernel32, User32 and GDI32 and others, So when a process need to do activity inside the kernel it use system calls and bunch of subsystem DLLs , and they mostly call another system wide DLL called NTDLL.DLL which implement the native windows API which has several uses one of which is to transition the process to the kernel mode into the system dispatcher which will try to implement the system service that was requested.

B. Fundamental to System Performance

Performance represents a crucial part of the functional needs. Ensuring that wall-clock time (the time taken to perform a job) requirements are met the control over the entire application and hardware stack. And that task has possible risks and maybe constrain the size of the system. Building fast and perfect software is a difficult task due to many reasons like the modern systems and typical performance engineering mistakes which makes the performance analyst in a tough challenge and open the aspect on this area. So, a performance issue is a concern that lift by the end-user, it has multiple symptoms like hang, crash, bottleneck, Resources leaking, software aging, fragmentations, and zombie processes. This concern represents a function of user insight since not all the issues reported through end-users could be indicated as a performance problem, and the valid ones could be like launching

an application could take a long time, system acting slowly, application crash, not responding software, system hang, etc.

It is important to know that finding a performance issue is not the main problem, especially on complex systems like Windows, because there could be many performance issues, and the real job is to identify and quantify which issue is matter the most [1]. So, for some issues finding the root cause factors could be requiring a mutual effort from more than one team, and since the need to use tools that perform sampling (taking a sample of quantifying to appear a coarse picture of the resource usage) that method of using tools represent what called profiling, and the affective visualization of resource profiles like Memory is a flame graphs, which can help the analyst to find more performance meet that any other tool, after metrics, because they not only show the memory issues but also the footprint of memory that a process leaves behind. Also, Memory performance issues can be analyzed by finding excessive CPU time in memory allocation functions (malloc ()), with the code path that across to them without releasing them, because there could be discording for how much there are allocations in the code compared with the deallocations, and since the deallocation part is on the responsibility of the developer, then the existence of the unbalance in those system calls there could be a significant impact on the performance.

II. Experimentation

The Experimentation tools which are a part of observability tools most of which are benchmarking tools, the applying of synthetic load to the system, and measuring its performance could be used through these benchmarking tools [1]. In Microsoft, there are multiple tools for Windows systems that it conducted to load and analyze the performance of windows activities (some of the important tools listed in Table 1) one of the Sysinternals tools is the command-line utility named testlimit.exe that could be used to make the system work out of memory in multiple scenarios [10] it could be used to the stress-test system and or/application by simulating low resource conditions for multiple resources like memory, handles, processes, threads, and other system objects. Also, simulating load manually by loading big file size to a 32bit application (e.g., notepad, outlook) to make it unresponsive and that a good indicator of a performance issue. A system utility called fsutil.exe could be used to create files with the needed size. Creating 600MB file as below to load 32-bit process**:**
fsutil file createnew <filename> < filesize>

Table 1.1: Some of Microsoft tools needed for Windows systems and applications [11]

Name	Descriptions
Assessment and Development Kit (ADK)	It is A package that has the tools needed to customize Windows images for large-scale deployment and to test the quality of the performance of a Windows system.
Windows Performance Toolkit (WPT)	It consists of performance monitoring tools that produce in-depth performance profiles of Windows operating systems and applications
Windows Debugger (WinDbg)	It is a kernel-mode and user-mode debugger that included in Debugging Tools for Windows
Sysinternals	Utilities that help IT and developers to manage, troubleshoot, and diagnose Windows systems and applications

Event tracing for Windows ETW

Microsoft Windows is a complex operating system with a lot of moving parts like services, and kernel components at the same time, so Microsoft build the ETW mechanism to understand things such as what is going on at every point of time, why things not going as it supposed to. So, ETW tries to answer those questions. It is important to trace low-level information like the stack that contains rich information about the internal activities like

functions, how much time it spends in the memory usage and other resources and system calls (the interface between a process and Windows kernel) and its anomalous could be corresponding CPU, Memory insufficient allocation and lower I/O throughput. It is important to conduct Event tracing for Windows (**ETW)** which is a better mechanism for root cause analysis to trace and log Windows events that could be raised by user-mode applications and kernel-mode drivers [12]. It comes with very low overhead, so it is possible to get thousands of events per second using ETW and still not overwhelm the Windows system, by mean the CPU and I/O performance should be very good, that is one of the goals of building this infrastructure. And to implement that ETW installs (**WPT**) which is the parse tool of **ETW**, which contains both Windows Performance Recorder (**WPR**) and Windows Performance Analyzer (**WPA**) [13]. As a summary there are several components [2]:

- Providers: the object that generates events for ETW (any recordable activity can be an event), there are more than thousands of providers inside Windows.
- Controllers: allow us to create a session (identify a set of providers and configure it) to enable a set of providers into that session and then to start/stop the session, so these providers send events, to be gathered in some internal ETW mechanism using buffering techniques than its information provided to the consumers and to get the file that targets all these information, this is event trace log file with the extension (.etl), then can take the trace file to analyze using variable tools like WPA of WPT that mentioned above.
- Consumers: it is an application that reads trace files or listens to active trace sessions and there are two ways for that real-time and offline.

So, at WPR could select the needed profiles to be recorded, CPU usage, Disc I/O activity, heap usage, and VirtualAlloc usage. Several processes used to load the windows OS and such as the testlimit.exe . The WPR to trace the events is used as shown in the below Figure 1.

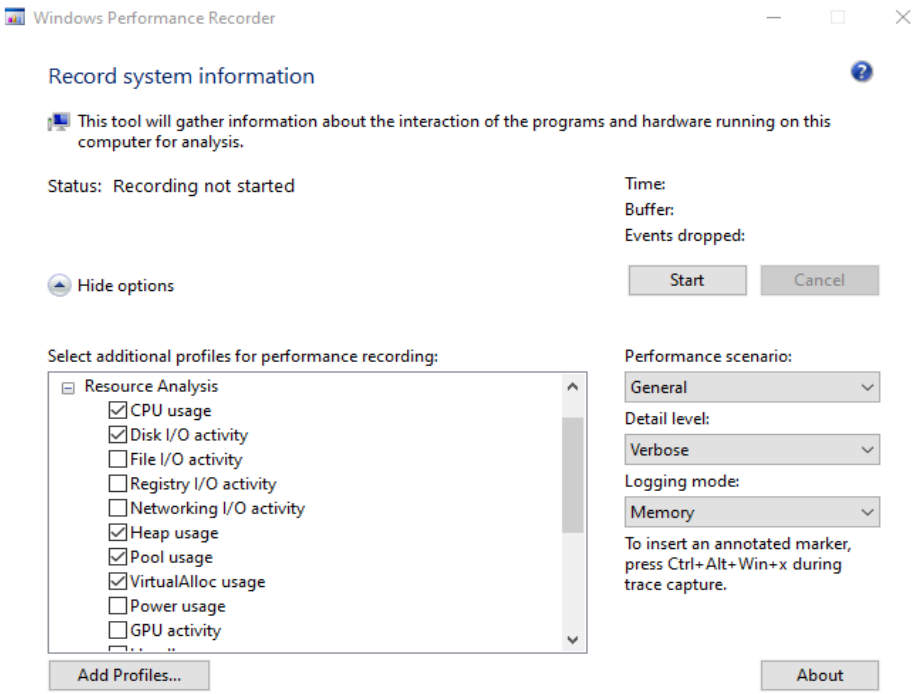

Figure 1: WPR with selected profiles

II. Analysis Methodology

After getting the .etl file from logging windows trace using WPR, the using WPA to analyze the collected events. In this scenario analyzing the workflow and assessing the affecting loading a very large file on windows 32-bit process. This study proposed the below method diagram to trace and Znalyze the system-wide events traced by the WPR:

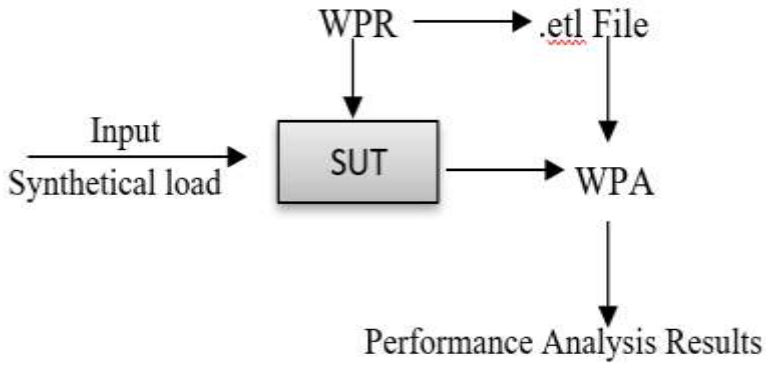

Figure 2: Method diagram to analyze events trace

The synthetical load that is used to simulate by the testlimit process to make the system under test (SUT) while recording the system internal activities with WPR, the important thing is to make the performance issue(s) symptoms happens inside the trace interval, to be analyzed then in WPA. In WPA there are multiple areas and graphs, and to start root cause analysis of the performance issues (unresponsive case) at first, it is important to indicate which area of interest in multiple metrics is interested to minimize and demonstrate our analysis, and according to that should check each area, analyze it, and give it area symbol. as below:

A. CPU Usage

One of the key terms for system performance is Utilization, which it is calculating how busy a resource is, depending on how much time in each interval a resource servicing work. At WPA starting from **Computation** section by drag and drop "**CPU Usage (Sampled) – Utilized by process**" and from the graph in the below Figure 3.

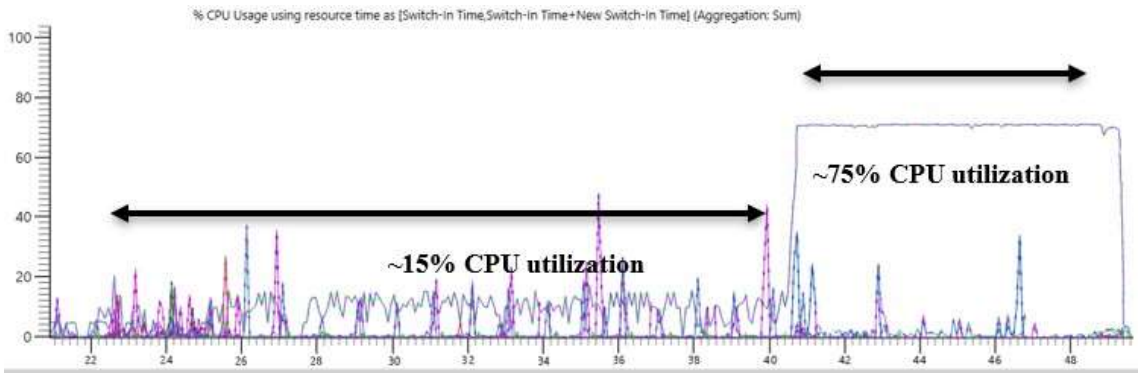

Figure 3: CPU Usage behavior during the system unresponsive (delay)

At the time interval ˜21 to ~41 the CPU utilization is under 20% of 0-100% scale, this area could be labeled as **(A_1)**, and after that interval, it jumped up to around 65% of utilization till ˜49 seconds, and that area of interest could be labeled as **A_2,** from that behavior there is something makes CPU busy at the area **A_2**, because the utilization should be at 100% scale of the CPU or any resource.

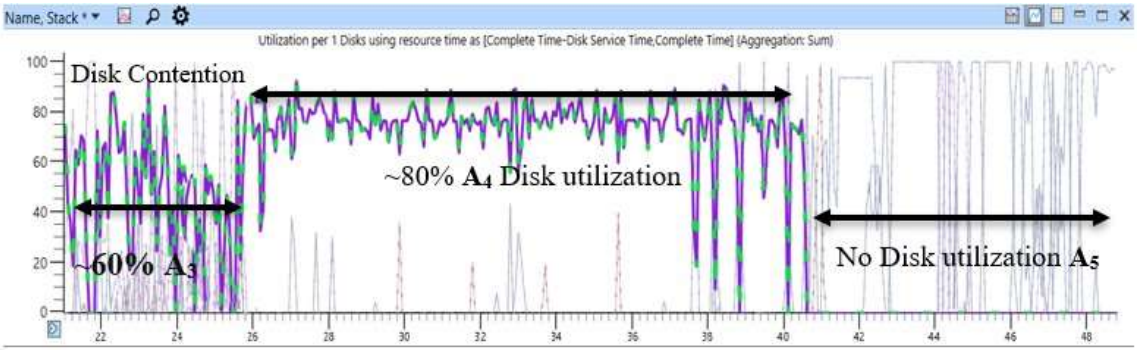

Figure 4: Disk Usage Utilization state during system delay

As a summary of CPU usage, the CPU is utilized through **A_1** neither **A_2**, as it should be 100% saturations for full utilization. So, it is important to see what Windows do in these two areas, hence next will go to the Disk Usage to see if Windows get well of Disk's throughput. But this is classic behavior of CPU underutilization caused by lack of parallelization with using of 3 out of 4 cores of CPU in the code like that maybe the source code of the affected process could be run on one logical processor at a time, and there is an improvement opportunity of CPU understanding by parallelizing CPU usage in **A_2**, which could increase to the 100% of utilization and (~ 0.35 seconds) of not benefiting from parallel activity, as below:
CPU activity = [(49.5s – 40.5s) – (~0.25)] / 3 cores = **~ 2.91 seconds**

B. Disk Usage and I/O contention

As we did at CPU usage metric in WPA to investigate in Disk IO, from **Storage** section drag and drop the "**Disk Usage – Utilization by process, pathName, Stack**", and from the below graph in Figure 6. there is contention (multiple processes try to access the shared resource Disk I/O) at the time interval ˜21s to ˜26s and that contention makes the Unresponsive process get ~60% utilization of the Disk bandwidth. Then this area of interest could be labeled as **A_3**. After that interval time, the unresponsive process gets utilization of Disk Usage by itself alone and that causes disk utilization to grow from ~60% (the contention in **A_3**) to ~80% of the Disk I/O bandwidth. Let be labeled that as **A_4**, so, there is an improvement opportunity for disk utilization at that A_4 as below.
Disk utilization A_4 = ~14s * 0.20 = ~**2.8 seconds**
That means it could speed and end the processing ~20% faster. Also, its utilization improved through ~80% - ~60% = 20% of unused leaving capacity. At the end of this area, there is no disk usage by the unresponsive process, and it is possible to make that area of no disk utilized as **A_5**.

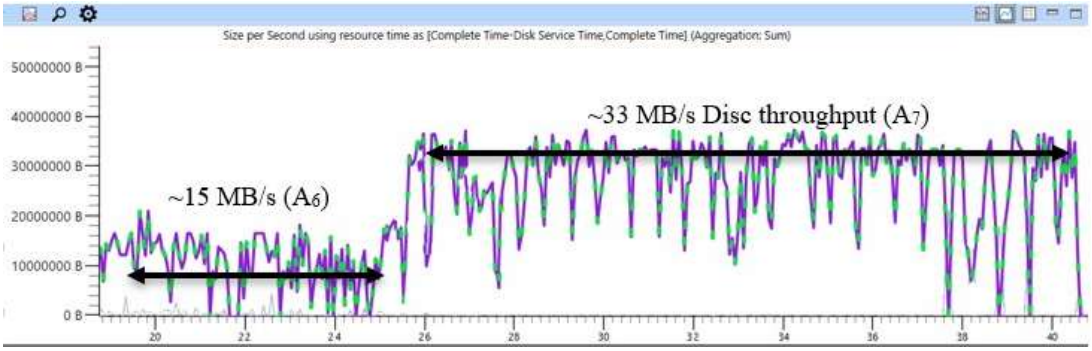

Figure 5: Throughput of Disk I/O usage

From the above analysis, the throughput of the Disk activity increased from **A_3** to **A_4** and end at **A_5.** and to see that could drag and drop the instance of the **Disk Usage** at WPA by the "**Throughput by Process, IO Type**" as shown in the below **Figure 5.**

The size of the process that makes reading service is (~15 MB) let called it area **A_6,** which it is lower than the same process (~33 MB) let it be the area **A_7**, it could remember at **Figure 6.** within **A_3** the leak process has contended with multiple systems and or user processes that trying to share the Disk IO, where at **A_4** no contention exists with other processes, and that what makes the reading utilization increased to ~33 MB, then it could get the average of the throughput at **A_6** as below:

$AVG_{throughput}$ at **A_6** = (~33 MB/s) / (~15MB/s) = ~2.2 sec

The $AVG_{throughput}$ at **A_6** is around ~2.2 times smaller than **A_7**, and without the contention, at **A_3** it could have a lower delay as below:

$AVG_{throughput}$ at **A_6** = (~6.5) / (~2.2) = ~**2.95** sec

that means without contention at sub-region **A_3** it could have ~2.95 seconds delay instead of ~6.5 seconds, which indicates that we lost around ~**3.55** seconds due to existing the other processes trying to access the Disk I/O (contention with testlimit.exe) at the **A_6.** So, the investigation in the Disk I/O contention of the area **A_3** showed that identify a slowdown about ~2.95 seconds, and there is an improvement opportunity to get as much as **~3.55 seconds** by reporting the existence of the contention at the area **A6**.

Results

As a summary of the above metrics which are CPU usage at the area (**A_2**), Disk usage at (**A_4**), and I/O contention at (**A_6**), it could speed up that scenario by speeding up the behaviors as much as declared in the below Table 2:

Table 2: Metrics Observed behavior and the Improvement opportunity

System Metrics	Area of Interest	Time Interval	Period in sec	Improvement Opportunities
CPU Usage	A_1	~ (40.5s – 49.5s)	~9 sec	~2.91 sec
Disk Usage	A_4	~ (26.5s – 40.5s)	~14 sec	~2.8 sec
Throughput	A_6	~ (21s – 26.5s)	~6.5 sec	~3.55 sec
Sum		**~ (21s – 40.5s)**	**~29.5**	**~9.26 sec (~30%)**

C. Memory Usage

To add Memory usage activities during the scenario in WPA from **Memory** group, add **Virtual Memory Snapshots graphs to the analysis**, add it under the Disk usage graphs to understand the whole scenario, as shown in Figure 6.

From the Figure 6 graph which is related to virtual memory snapshot, observing from the time interval ~21s to ~26s that there is slow allocation on memory than time interval ~27s to ~41s, then there is a significant drop (deallocation) in the memory exactly at the time ~41.5s, and depending on the methodology of the analysis it could separate the above graph's areas into $\mathbf{A_8}$ = ~21s to ~26s (low allocation), $\mathbf{A_9}$ = ~27s to ~41.s (High allocation), $\mathbf{A_{10}}$ = ~41.5s (the significant deallocation) and $\mathbf{A_9}$ = the constant footprint of memory after deallocation.

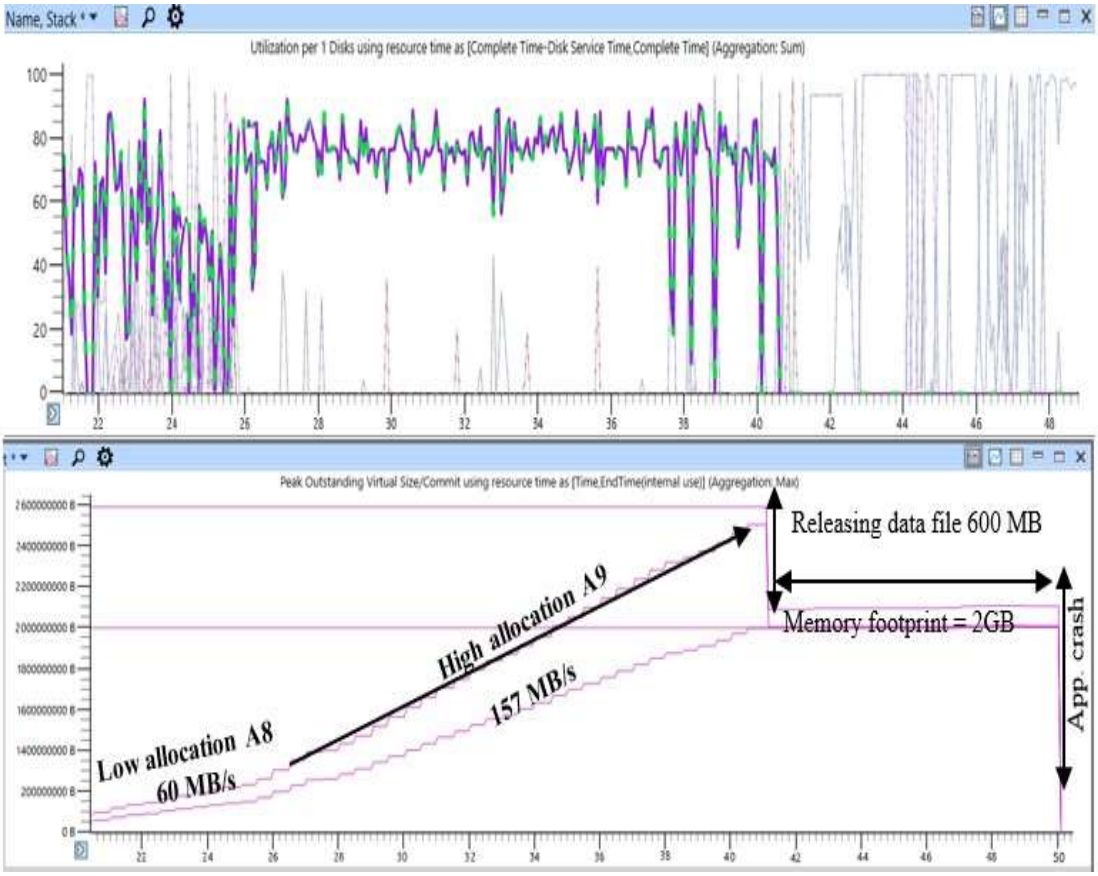

Figure 6: Adding the entire Memory usage graph of the process

Memory leak root cause analysis

To detect the root cause that keeps allocation on the memory, then it is important to drag and drop an internal metric called < **VirtualAlloc Commit LifeTime Snapshot>** section from the memory section inside the WPA. After applying customizing on the chart and graph with the related synchronize table on the same analysis region, adding the commit stack and impacting stack to get the best view for analysis, and after digging into the call stack column of the code path, we get the below results:

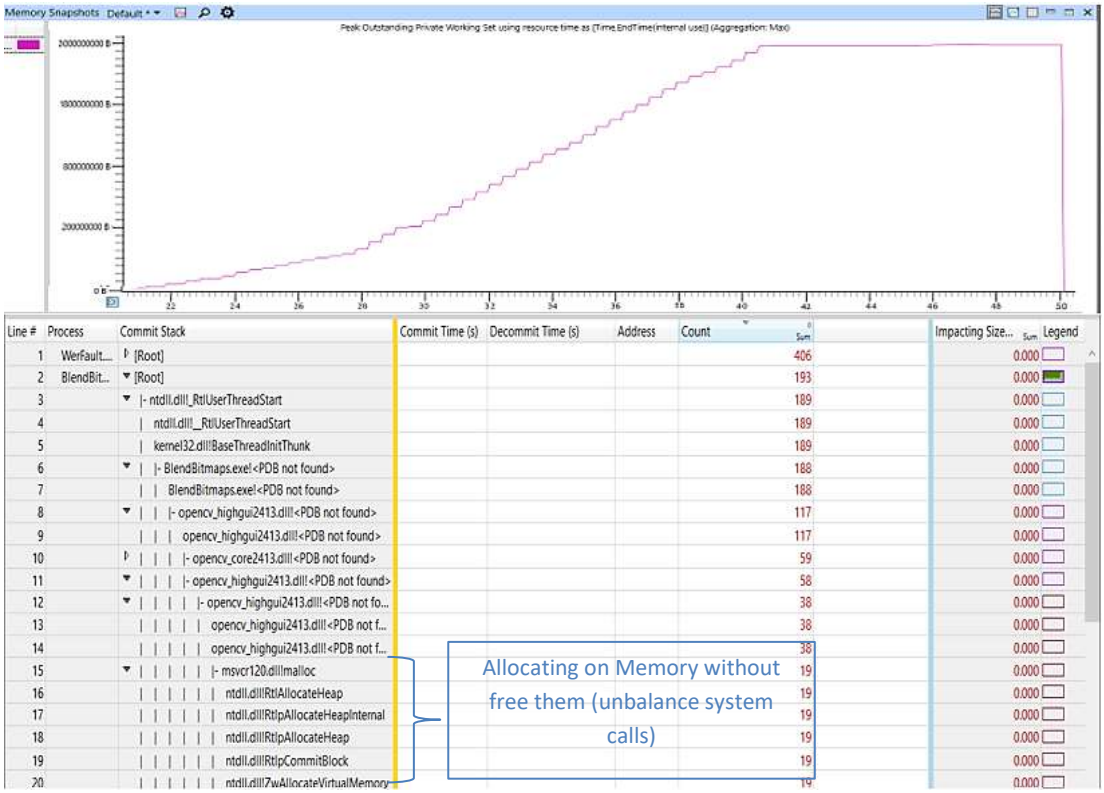

Figure 7: Digging into the code path stack to find out the reason of the abnormal behavior

By the way, **call stacks** help to control and manage the complexity of the code path, it shows important information for the analyst like function calls, methods, related threads that are used in a context, etc. From the above Figure 8. And while digging into the call stack column, it shows that the highgui2413.dll function library has been used for that allocation, and then keep digging into its stack, it shows that it is called the malloc system call for deserving and touching memory and it keeps growing till consuming all the available bytes in its commit virtual size, the reason that shows it never used deallocate or dealloc system call after completing its reading task.

Results and Discussion

To estimate the increase in memory allocation it is important to calculate the speed at the area $\mathbf{A_8}$ which is ~60MB/s that comes from ~300MB (allocated during the area) / 5s of the $\mathbf{A_8}$. And at the area $\mathbf{A_9}$ which it is 2.2G / 14s which equates to ~157MB/s as the rate of the allocation at the area $\mathbf{A_9}$. The slow memory allocation process rate that happened at $\mathbf{A_8}$ which it is the same as $\mathbf{A_3}$ (the area of I/O contentions) and hence observing the reason for the low memory allocation which impacted by the I/O contention $\mathbf{A_3}$, So the I/O contention slowed the test limit process that becomes unresponsive symptoms. And the higher allocation area $\mathbf{A_7}$ which at the same time as area $\mathbf{A_4}$ increased memory allocation due to the absence of the contention in this area.

The entire size that Memory allocated is around ~2.6 GB and the deallocation at ~41.5s is ~600MB/s. So, from that point, it is clear to see that the memory used 2 GB to read 500MB data file, and after completing the read service it dropped the size of the reading file and deallocate it with deserve with the memory footprint as represented in $\mathbf{A_9}$ which it is 2 GB. The ambiguity could be a limitation of this study for further investigation in memory usage to know the reasons for this memory footprint deserving, which could be solved through looking into the source code or using the ETW activities to investigate the entire stack.

Conclusion

As a summary, while increasing the synthetical load using testlimt.ext, when a 32-bit process tries to read data with size ~600MB it peaks up to ~2.6GB instead of just 600MB of the file size with unresponsive symptoms, and after reading that file the process released the 600MB (the size of the file as in Figure 8) back to the operating system and leave it with ~2GB of constant leaking as a memory footprint. During the time interval of I/O contentions (the area $\mathbf{A_3}$), there is slower throughput which causes a slow in memory allocation at the area $\mathbf{A_6}$ of the Memory Usage graph. So, the memory utilization is not used in an efficient behavior in that scenario, despite there being 8GB of Main Memory is installed in the system configuration of Windows 10. Even when there is is not enough RAM then the system will be depending on the page file taking from the working set of the process to the page file to free up the RAM and deserve the hungry 32-bit process, and that may lead to hanging the page file activity. The call stack column helps us to detect the abnormal behavior by highlighting the related syscalls which keep allocation on the memory without releasing. According to the CPU and Dis Usage metrics, and improvement opportunity on the resource is utilization has been conducted by calculating the losing time for each other, which after adding their values it shows that it could minimize the time down into ~9.26 sec instead of ~29.5 sec, as shown above in Table 1, which show around ~30% of utilization.

References

[1] B. Gregg, *System Performance Enterprise and the Coud*, Second Edi. 2020.

[2] I. Park, "EVENT TRACING FOR WINDOWS : BEST PRACTICES," *http//www.cmg.org EVENT TRACING Wind. BEST Pract. Microsoft Corp.*, 2004.

[3] M. Kubacki and J. Sosnowski, "Exploring operational profiles and anomalies in computer performance logs," *Microprocess. Microsyst.*, vol. 69, pp. 1–15, 2019, doi: 10.1016/j.micpro.2019.05.007.

[4] J. Zhu *et al.*, "Tools and Benchmarks for Automated Log Parsing," *Proc. - 2019 IEEE/ACM 41st Int. Conf. Softw. Eng. Softw. Eng. Pract. ICSE-SEIP 2019*, no. March 2020, pp. 121–130, 2019, doi: 10.1109/ICSE-SEIP.2019.00021.

[5] K. S. Noori and A. A. Fahad, "Factors affecting application launch time with android OS," *Iraqi J. Sci.*, vol. 61, no. 7, pp. 1791–1797, 2020, doi: 10.24996/ijs.2020.61.7.28.

[6] K. S. Noori and A. A. Fahad, "Monitoring and enhancement of mobile system performance," *Iraqi J. Sci.*, vol. 61, no. 9, pp. 2418–2425, 2020, doi: 10.24996/ijs.2020.61.9.28.

[7] J. Milosevic, M. Malek, and A. Ferrante, "A friend or a foe? Detecting malware using memory and CPU features," *ICETE 2016 - Proc. 13th Int. Jt. Conf. E-bus. Telecommun.*, vol. 4, no. Icete, pp. 73–84, 2016, doi: 10.5220/0005964200730084.

[8] P. K.Sahoo, R. K. Chottray, and S. Pattnaiak, "Research Issues on Windows Event Log," *Int. J. Comput. Appl.*, vol. 41, no. 19, pp. 40–48, 2012, doi: 10.5120/5650-8030.

[9] T. Pantels, "Touch Response Measurement , Analysis , and Optimization for Windows * Applications," pp. 1–15, 2014, [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/touch-response-measurement-analysis-and-optimization-for-windows-applications.html.

[10] "Testlimit - Windows Sysinternals | Microsoft Docs." https://docs.microsoft.com/en-us/sysinternals/downloads/testlimit (accessed Aug. 06, 2021).

[11] "Windows Documentation | Microsoft Docs." https://docs.microsoft.com/en-us/windows/ (accessed Mar. 24, 2022).
[12] "Event Tracing for Windows (ETW) - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/event-tracing-for-windows--etw- (accessed Aug. 07, 2021).
[13] "Windows Performance Toolkit | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/test/wpt/ (accessed Aug. 07, 2021).