Sample excerpt from Chapter 5: Identifying Resource Usage Based on Regions of Interest

…

Region R2: CPU underutilization

1. In the previous section, we've looked at a case of disk underutilization in region ***R4***, which was caused by synchronous interleaving of CPU and disk activity initiated by Notepad. In this section, let us look at another case of a key resource being underutilized. Namely, the ~50% CPU utilization that we saw in region ***R2*** in **Error! Reference source not found.** (shown again for your convenience in Figure 0.1).

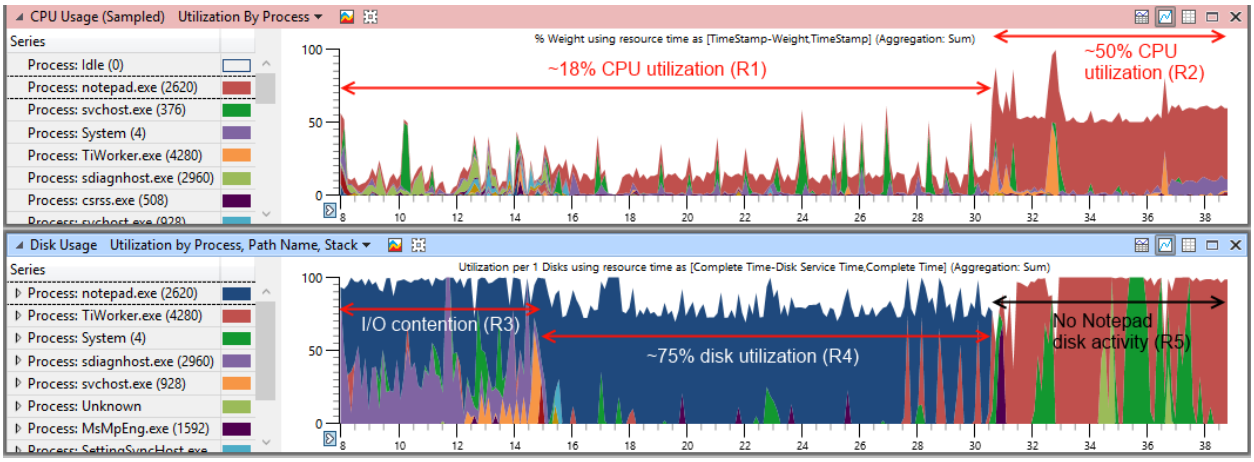

Figure 0.1 CPU and Disk Usage during Notepad UI delays

2. Let us start by zooming in on the time region ***R2***. Since there was no disk activity from Notepad at this time, we only need to look at its CPU usage. To that end, let's take a look at both the compound holistic CPU usage view as well as the one filtered to just `notepad.exe` process. Note that you can either add two instances of the `CPU Usage` graph from Graph Explorer or just duplicate the graph within the view using the `Duplicate graph` option in right click context menu (dragging and dropping a graph by the caption with pressed `Ctrl` works too, as expected). We'll want one of these instances to show additive CPU usage (using stacked line graph mode) and another to just show us CPU usage of `notepad.exe` process (using filtering). The resulting view should look as shown in Figure 0.2.

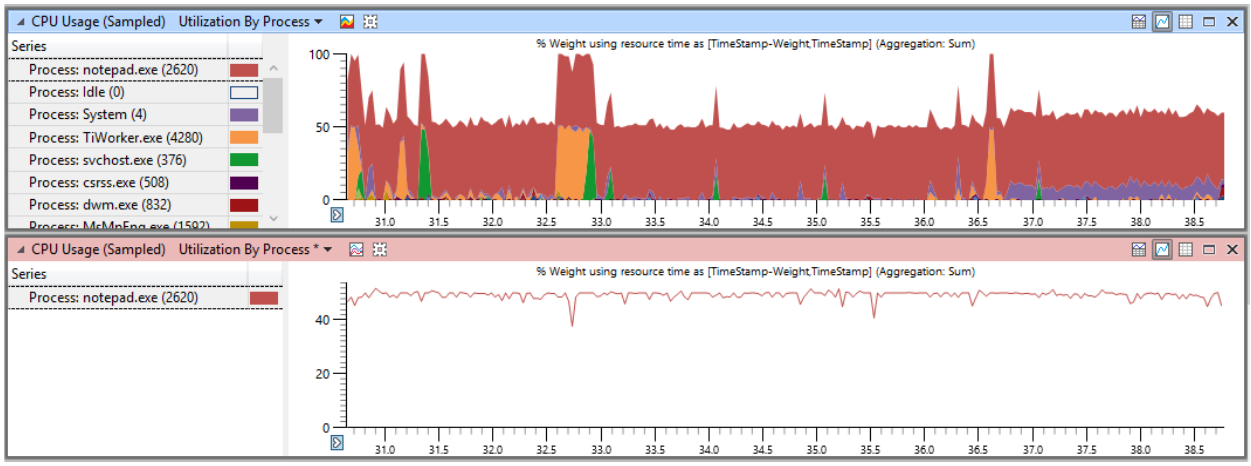

Figure 0.2 CPU usage (holistic and notepad.exe) in region R_2

3. First thing you notice is that Notepad CPU usage is pegged steadily at 50% of CPU. One could thus argue that if Notepad used CPU fully here, it would complete its work twice faster.

4. Why 50% specifically, you may ask? As we'll see in Chapter 7, CPU utilization of activities trying to keep the CPU busy tends to have very distinct levels: 50%, 25%, 12.5%, etc. What do all these numbers have in common? They are inverses of powers of two: 1/2, 1/4, 1/8, etc. It so happens that modern systems commonly come with multiple processors, and the number of processors is typically a power of two (as you probably know, and as we'll discuss in Chapter 7, these processors can either be distinct CPUs or cores on a given CPU, which is why we typically refer to them as **logical processors**). If a given activity is utilizing one of these processors, you'll see it appear as pegged at one of the mentioned percentage levels, depending on how many processors that system has.

***NOTE**: Not all CPUs have the power of two number of cores. There are products on the market that have core counts that are not power of two.*

5. Let's see what kind of a CPU this was, by examining a very useful feature in WPA, called `System Configuration`, accessible via the `Trace` menu, as shown in Figure 0.3.

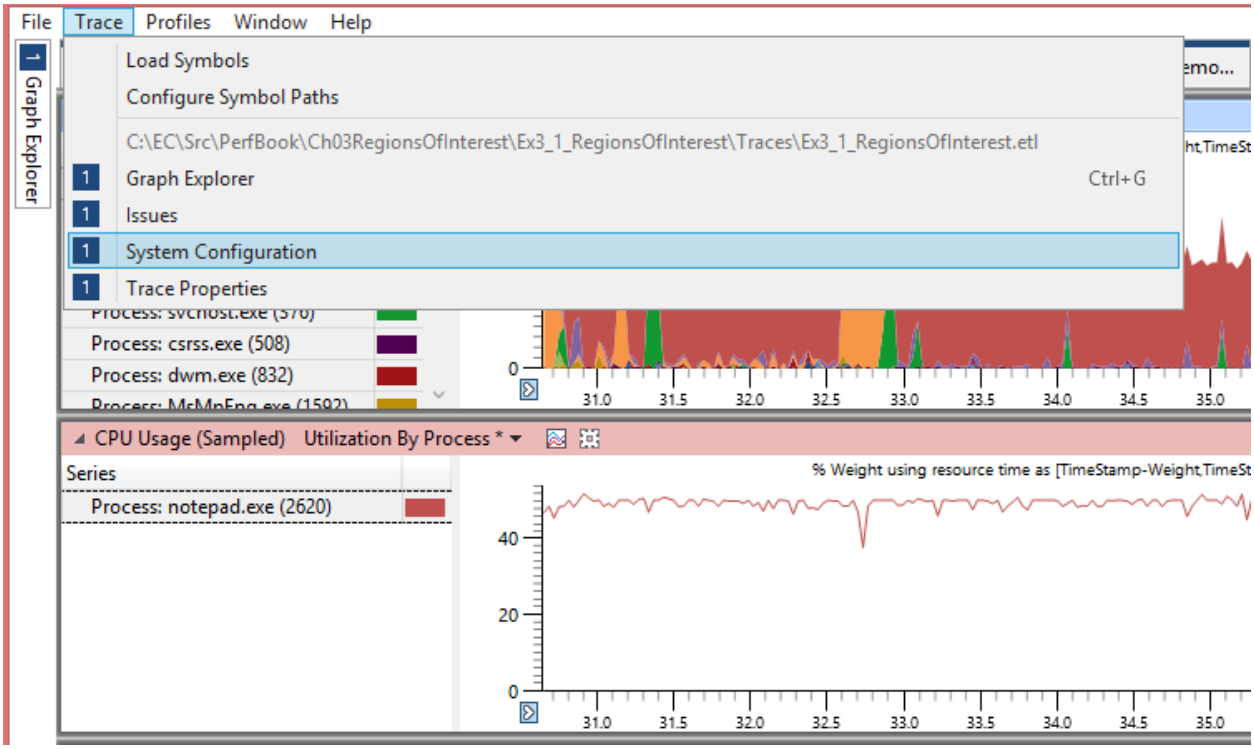

Figure 0.3 System Configuration menu option

6. WPA will then open a separate view with the summary of key properties describing the system that the currently opened trace was collected on. It will look as the view shown in Figure 0.4.

General
Traces
Trace Statistics
Storage
Display
Network
Services
IRQ
PnP

Line #	Configuration	Value
1	OS Version	6.2
2	Build	9200
3	System Manufacturer	Dell Inc.
4	System Product Name	Latitude XT2
5	BIOS Date	05/03/2011
6	BIOS Version	A12
7	Computer Name	PERFBOOK
8	Domain Name	
9	Product Name	Windows 8
10	Build Lab	9200.16628.amd64fre.win8_gdr.130531-1504
11	Processor Name	Intel(R) Core(TM)2 Duo CPU U9400 @ 1.40GHz
12	Number of Processors	2
13	Processor Speed	1395 MHz
14	Hyper-Threading Enabled Processors	0x0000000000000000
15	Memory Size	5072 MB
16	Page Size	4096 Bytes
17	Allocation Granularity	65536 Bytes
18	Supported Power States	S3 S4 S5
19	Boot Drive	Disk 0 - Drive C - NTFS

Figure 0.4 System configuration of system used to repro Notepad opening data.txt

7. As the system configuration view for this trace shows us, our repro system had two logical processors, which explains why Notepad CPU usage was pegged at 50%.
8. This is a classic case of CPU underutilization caused by lack of parallelization in Notepad's code: i.e. Notepad's code could only run on one logical processor at a time. To improve Notepad's use of CPU resources, it would need to take advantage of multi-threading (another concept that we'll discuss in Chapters 6 through 9 9).
9. Notice that while Notepad's CPU usage is relatively constant, total CPU usage does sometimes spike from ~50% to ~100%. One such prominent case is between 32.5s and 33s, as shown in Figure 0.5. Let's call this region, ***R*$_7$** and discuss it in the next section.

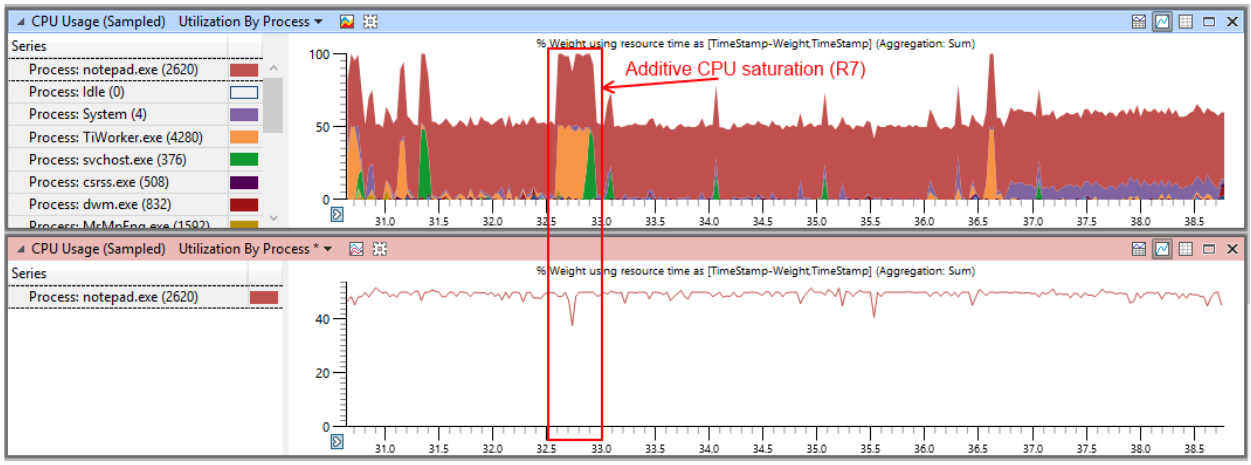

Figure 0.5 Additive CPU saturation region *R*$_7$

10. Here is another potential **improvement opportunity**: our investigation of CPU underutilization showed that by parallelizing CPU activity in region ***R*$_2$** we could gain as much as [(39s - 30.5s) – (~1s of concurrent activity that isn't likely to benefit from Notepad's parallelization)] / (2 processors) = **~3.75 seconds.**

Region R7: Additive CPU saturation

11. Let's next take a closer look at region ***R*$_7$** to better understand the nature of additive CPU saturation. This time, we'll use three different copies of the `CPU Usage (Sampled)` graph with three different presets:
    - One showing utilization by CPU (across all processors) called `Utilization By CPU`,
    - Another one by process (across all processes) called `Utilization By Process`, and
    - Finally one showing utilization by just the `notepad.exe` process (i.e. previous graph filtered to just one process).
12. We'll switch the first two graphs to use stacked line graphs to help convey the additive nature of CPU users. This view, zoomed to a time range between 32.5 seconds and 33 seconds (i.e. region ***R*$_7$**), should look as shown in Figure 0.6.

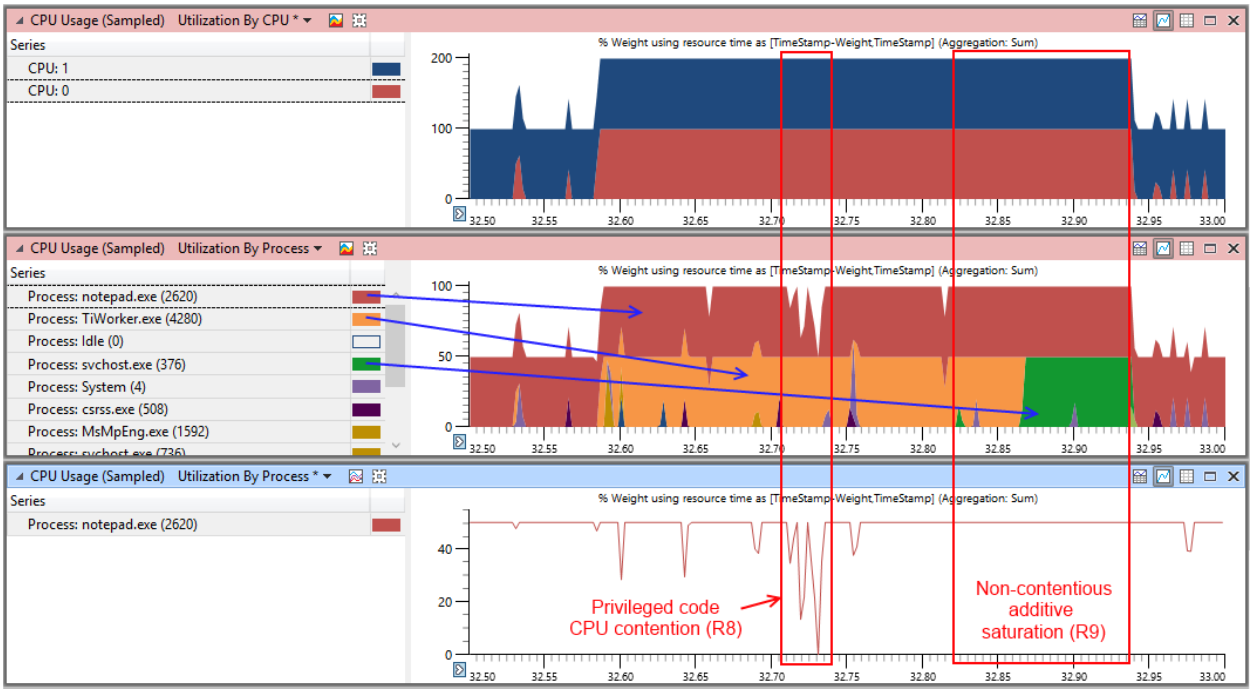

Figure 0.6 Region R_7 with sub-regions R_8 and R_9

***NOTE**: Note that, as we've mentioned previously, WPA does not use consistent colors across its graphs, so the color chosen for CPU `0` in the first graph has no logical relationship with the color chosen to represent `notepad.exe` activity in the second graph.*

13. Recall that multiple concurrent users of a typical single storage device tend to cause I/O contention, slowing down each other's execution. This is because most consumer storage devices aren't particular good at concurrently servicing multiple requests, and instead tend to queue them up. This has been a lot more impactful with rotational devices, but SSDs have also been known to be affected by concurrent usage.

14. The same is not necessarily true with CPUs. As long as there are more processors available than the number of concurrent activities, each activity gets to run on a separate processor and they typically do not interfere with each other.

***NOTE**: Even when there are enough logical processors for each concurrently executing activity to run in parallel, they can still impact each other through various secondary effects (e.g. cache invalidation, memory bus contention, etc.). Discussing these is beyond the scope of this introductory text.*

15. Let's take a closer look at the two sub-regions of region R_7, as shown in Figure 0.6, namely:

    - R_8 that represents sub-region between 32.70 seconds and 32.75 seconds, where `notepad.exe` CPU usage dipped all the way to zero.
    - R_9 that represents sub-region where `notepad.exe` CPU usage was at a steady 50%.

16. Specifically, let's contrast regions R_8 and R_9. In both of these regions, we see that `TiWorker.exe` is running concurrently with `notepad.exe`. In fact, in R_9, `svchost.exe` joins the party, replacing

`TiWorker.exe` around ~32.87s. With all this contention, you might expect that Notepad's CPU usage would drop. You might even think that this is exactly why Notepad's CPU usage did drop in ***R*$_8$**. That said, you'll notice that no other processes' CPU usage increased noticeably during ***R*$_8$**, so something else must have stolen away these CPU cycles.

17. Also, interestingly enough, in ***R*$_9$**, despite the presence of an even greater number of concurrent CPU users, Notepad's CPU utilization didn't miss out a single heart beat and kept steady at 50%.
18. What overruled Notepad's usage of CPU in ***R*$_8$** was, in fact, the underlying system itself – i.e. the combination of the operating system and the drivers installed on it. As we'll see in Chapter 9, these activities are referred to as DPCs (Deferred Procedure Calls) and ISRs (Interrupt Service Routines). When we take a closer look at how the OS is performing its low level tasks, we'll show you how to visualize this type of CPU consumption.

***NOTE**: As you can imagine, a misbehaving driver can thus easily wreak havoc on an end-user system. Being an open platform, every Windows PC in existence has third party (i.e. non-Microsoft) drivers. Even before users start installing custom drivers on their shiny new machines these drivers have already been preinstalled on their systems. These include anything from web cameras to finger-print readers to latest graphics cards. Many of these drivers weren't even part of the original operating system image and were added by the OEM at the factory floor. Unfortunately, not all of these drivers come bug-free. While Microsoft has created a very thorough quality control process for inbox drivers, the quality control for third party drivers, including those certified through WHQL (a quality control process imposed on any driver that wants to be shared through Windows Update) may not be as rigorous.*

19. Given that each of the processes during region ***R*$_7$** (i.e. `svchost.exe`, `TiWorker.exe`, `notepad.exe`) was not utilizing more than 50% of CPU time, we can deduce that their execution was likely limited to a single logical processor at a time (recall that this system had two logical processors). Does this mean that these processes didn't impact each other?
20. As we've seen, there were three processes consuming CPU time with only two logical processors available, so there was likely some contention. At the same time, `notepad.exe` seems to have been getting its share of CPU time throughout the entire region ***R*$_7$**, unaffected by the other two processes. Had we addressed Notepad's CPU underutilization in this region by having it utilize multiple processors, the impact of CPU contention by other two processes on `notepad.exe` would likely become much more apparent on this particular system.
21. There are two key takeaways here:
    - Multiple concurrent CPU users do not necessarily steal CPU cycles from each other, and, unlike with most disks, don't necessarily result in contention (caveat: some disks can handle multiple concurrent accesses too).
    - Even when you have enough processors to satisfy all of the concurrently running processes, system itself may steal away from your resources (after all, system code and drivers also run on the CPU).

22. We will look more closely at the interplay of additive CPU saturation and the impact of privileged code execution in Chapter 7 and Chapter 9.

Exercise summary

Wow, that was quite a long exercise! Resource usage centric analysis has certainly proven to be a rich source of insights. The newly discovered understanding we've gained can be invaluable in finding how to improve it. Specifically, addressing I/O contention in region R_3, disk underutilization in region R_4, and CPU underutilization in region R_2, could speed up the scenario by as much as ~4.4s + ~3.75s + ~3.75s = ~11.9s, or approximately ~12s, a very sizable improvement. With the entire UI delay taking ~30.85s, shaving off ~12s would constitute an improvement of roughly 40%!

Note that some of these improvements are easier to tackle than others. Choosing the right remediation tactics when faced with multiple options requires thinking about the trade-offs and cost/risk/benefit analysis.

Of course, these improvements are merely fine tuning the existing approach that Notepad developers decided to take when dealing with loading large files. There is a morale to this story. While we could keep ourselves busy fine tuning identified shortcomings of the existing implementation, it can pay a lot more to take a step back and consider a different approach altogether.

By reexamining the overall design, and, perhaps, opting for a "sliding window" access model to the data file (which would only require loading as much data as is shown by the current window and, perhaps, a little more on the sides), we could further improve performance of loading large files by orders of magnitude. It's important to keep this in mind – sometimes, rather than investing thousands of dollars into repairing your old clunker, it's just best to look into getting a new car.

So far we've managed to identify numerous opportunities for speeding up Notepad's scenario of opening a large file by merely looking at its CPU and disk usage. Are there more issues, and resulting improvement opportunities, lurking in this trace though? In the next exercise we are going to look at another shared and limited critical system resource: memory.

With CPU and disk, if you use the resource you are typically taking the time away from the scenario. With memory, things are a bit more involved. Recall that memory capacity pressure results in paging, which can lead to I/O contention. Thus a large memory footprint on a system with low available memory can potentially impact nearly all scenarios interacting with that secondary storage device.

With the recent advancements in memory capacities on typical consumer systems, causing memory pressure due to functional requirements of common user scenarios is actually not very typical (short of having memory leaks in your code). And until a device runs out of memory, simply using larger memory allocations does not necessarily translate into slower execution.

Making a lot of allocations and deallocations, however, might very well impact the duration of your activities, potentially resulting in fragmentation issues.

***NOTE**: Ineffective allocation patterns can result in fragmentation of memory pools/heaps used to satisfy these allocation requests.*

Also, as inexpensive as a single allocation and deallocation instructions are, do enough of them and you'll suffer from the death by thousand cuts.

Another consideration can stem from the way memory gets accessed. Just like with secondary storage devices, some memory access patterns can be more efficient than others (this has to do with processor execution optimizations and caching).

***NOTE**: How you access allocated memory also oftentimes impacts duration of execution. Memory access patterns that result in CPU cache misses cause CPU to fetch data from memory, which can be orders of magnitude slower than accessing the same data from the cache.*

Of all of these considerations, memory capacity pressure is arguably the simplest to both reproduce and identify. In the next exercise we are going to show you WPA graphs you can use to review the memory footprint of the Notepad application.

The rest of these considerations require advanced analysis techniques, which are beyond the scope of this introductory text.

Without delving into these more advanced quirks of memory subsystem operation, let's review the overall memory usage of Notepad to see if there are any opportunities for improvement with regards to how it's using system memory.

***NOTE**: While CPU and disk contention are very common in typical user scenarios, we sometimes also see contention on memory due to cache coherency issues, when multiple processes with large actively used memory footprints are continuously invalidating each other's cache lines in L2/L3 caches. Since access to main memory is orders of magnitude slower than access to the CPU cache, this may have a material impact on overall performance. You would typically see this as increase in CPU usage, since memory access instructions end up "costing" a lot more cycles in those scenarios. Looking at memory contention is beyond the scope of this introductory text.*

Exercise 5.2: Focusing on Notepad memory usage within its UI delay

In this exercise we will leverage WPA's memory usage graphs to review Notepad's memory footprint. In doing so, we will again be looking for distinct patterns that help us identify regions of interest corresponding to specific activities within the trace.

1. To get started either:
    - Continue with the views we've built in the previous exercise if you've kept WPA open
    - Alternatively, repeat steps in Exercise 5.1 leading to **Error! Reference source not found.**.
    - Or, simply re-open the trace `Ex4_1_RegionsOfInterest.etl` from `\Ch04\Traces`, apply `\Ch05\Traces\Ex5_2_MemoryUsage.wpaProfile` using the `Profiles|Apply…|Browse…` menus, and zoom to `notepad.exe` UI Delay interval using the `UI Delays` graph at the top

2. Your view should look as shown (again) in Figure 0.7.

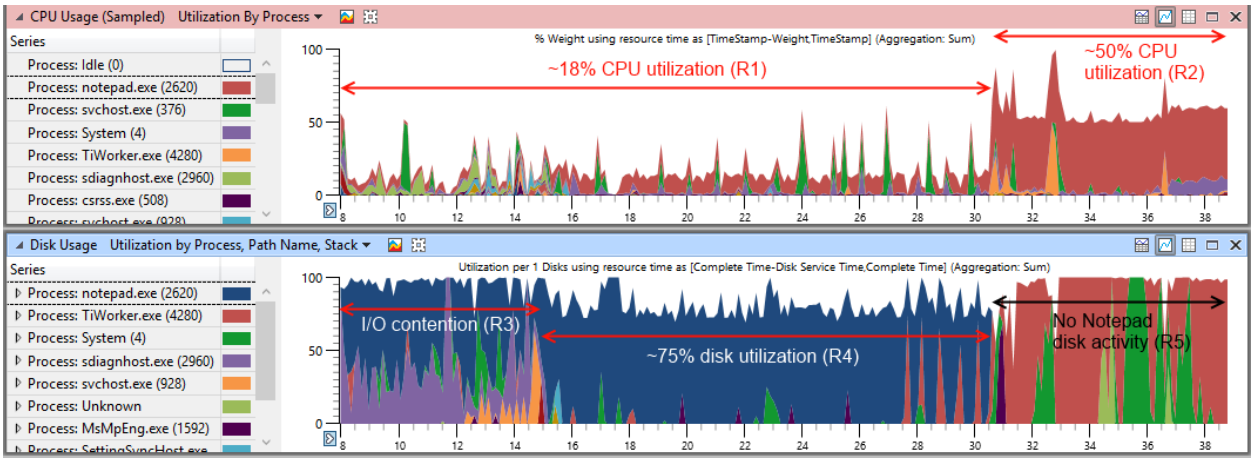

Figure 0.7 CPU and Disk resource usage along with corresponding regions of interest during Notepad's UI delays

Cross-resource CPU/disk/memory view

3. Let's add the holistic memory usage view in order to contrast what was happening with system memory usage as Notepad was opening the file. To do so, expand the `Memory` group in Graph Explorer and add `Virtual Memory Snapshots` graph to the analysis view, as shown in Figure 0.8.

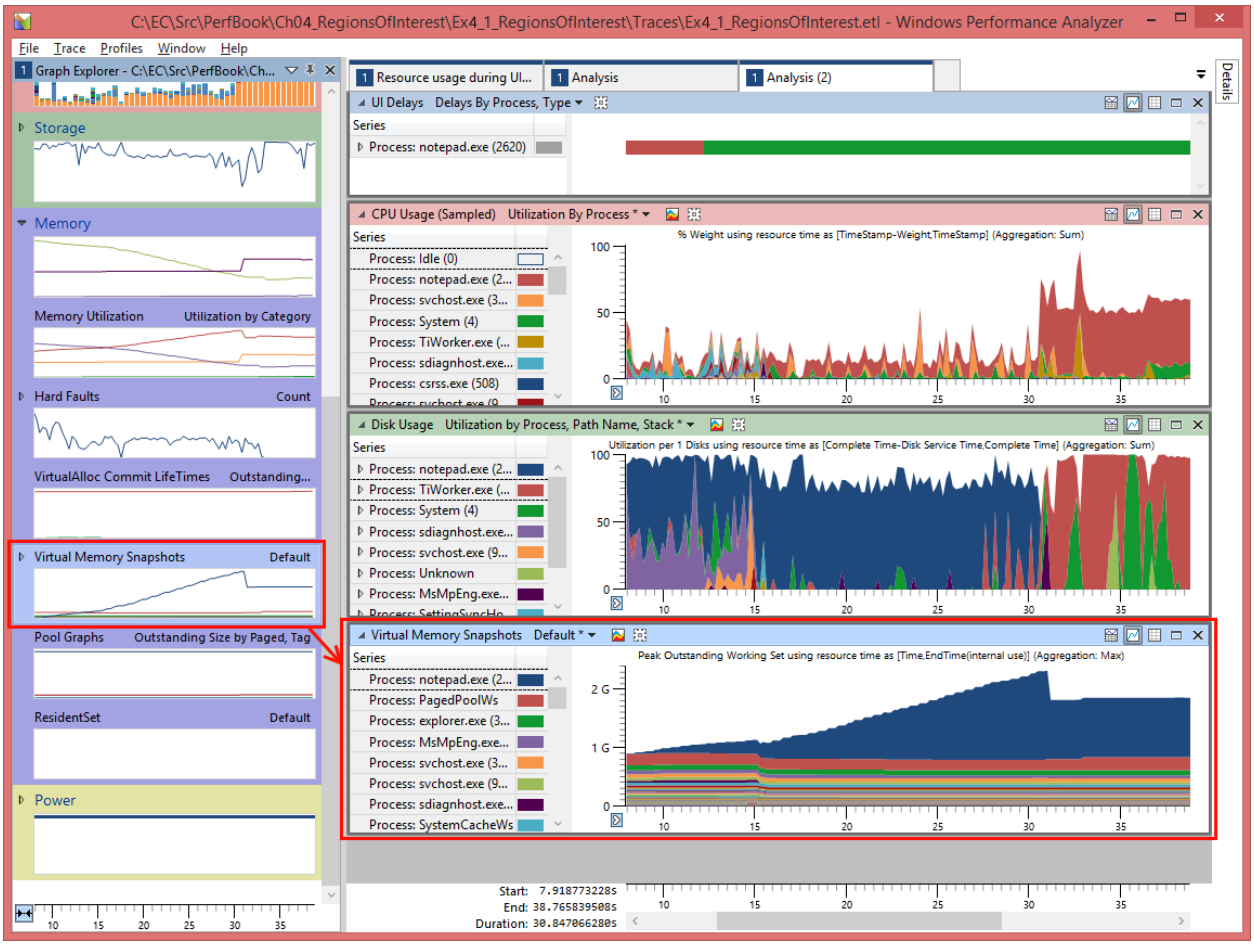

Figure 0.8 Adding holistic memory usage view during Notepad UI Delay

4. Let's now take a closer look at the memory usage graph and, specifically, the regions of interest within it, shown in Figure 0.9.

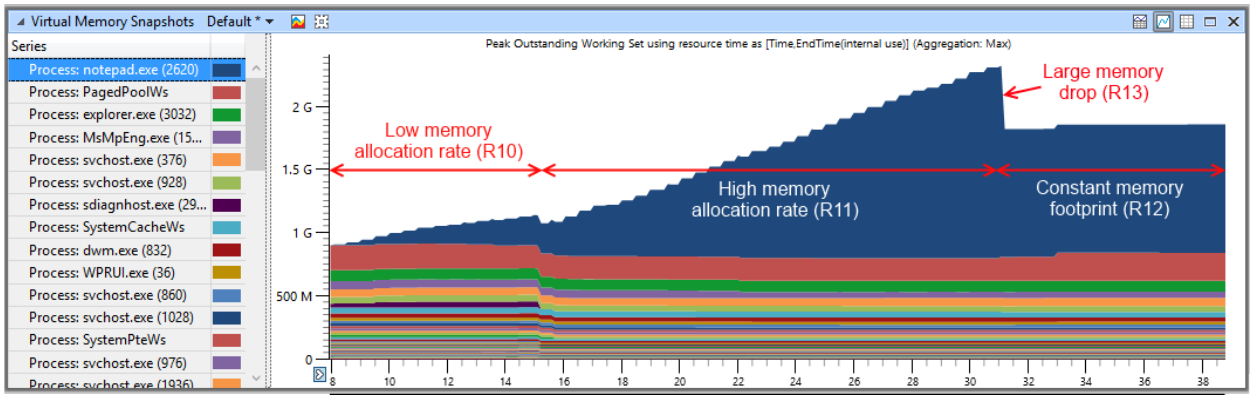

Figure 0.9 Memory usage regions of interest during Notepad's UI delay

5. Observe that from ~8 seconds to ~15 seconds Notepad's memory usage grew at a slower rate than from ~15 seconds to ~31 seconds. You can then see a significant drop in its memory usage at ~31 seconds, and from that point on, system memory usage held steady. This partitions the timeline into four different regions of interest, as shown in Figure 0.9, namely:

   - R_{10} = (~8s, ~15s): low memory allocation rate
   - R_{11} = (~15s, ~31s): high memory allocation rate
   - R_{12} = (~31s, ~38.5s): constant memory footprint
   - R_{13} = ~31s: Large memory drop

***NOTE**: R_{13} is a region of interest depicting a vicinity of a point in time representing a state transition (i.e. deallocation of memory).*

6. Since we are no longer interested in the contending activities, let's switch the `Virtual Memory Snapshots` graph to a line graph mode and then filter to just what `notepad.exe` process was doing on the system, as shown in Figure 0.10.

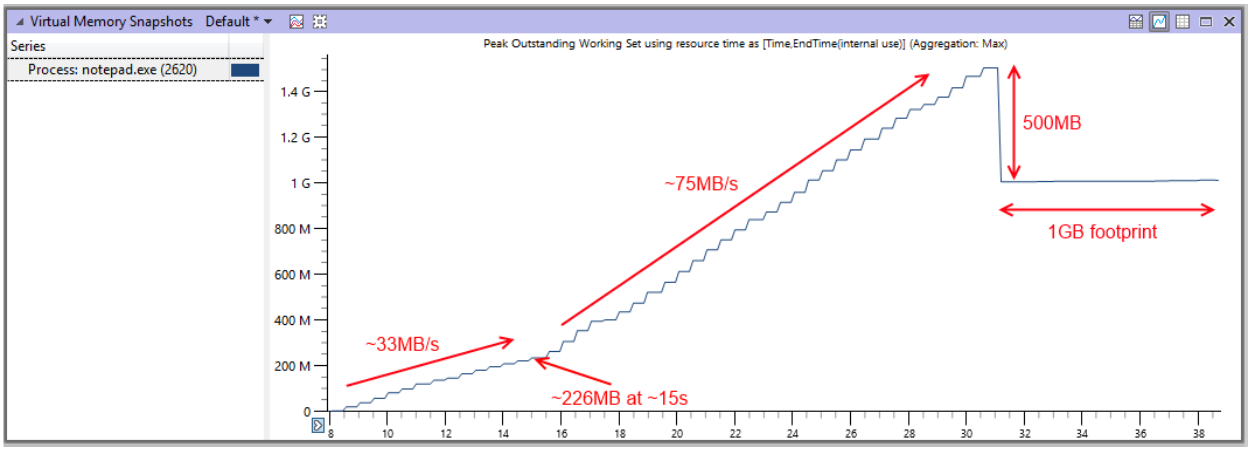

Figure 0.10 Memory usage regions of interest

7. Ouch! Looking at the vertical axis we can see that Notepad has allocated 1.5GB of memory to read in a 500MB file. Don't you think this is a bit inefficient? Well, at least it gave us 500MB back later at ~31s. Any guesses as to what Notepad was doing there?

8. Another interesting observation is that during the time region when Notepad was contending with other processes for disk time, its memory allocation growth was significantly impacted by that contention. To be more specific, in region ***R*$_{10}$** (same as disk I/O contention region ***R*$_{3}$**) the memory allocation rate was about 40% of the rate in **R$_{11}$** (same as disk I/O contention ***R*$_{4}$**). We can actually explain the speed up from ***R*$_{10}$** to ***R*$_{11}$** by making an educated guess that Notepad memory allocation rate is proportional to its read I/O rate. Given that I/O contention slowed down Notepad's ability to read in the file, Notepad also ended up allocating memory at a slower rate.

9. In fact, note that memory allocation growth rate was roughly three times the read throughput that we saw earlier. To estimate the memory allocation growth in region ***R*$_{10}$** you can zoom in to that region and observe that we've allocated 226MB during the initial 7 seconds, which equates to **~33MB/s**. Similarly, zooming into region ***R*$_{11}$** we can see that during these 16 second, we've allocated 1,437MB – 228MB = 1,209MB, which is about ~**75MB/s** allocation rate.

10. Recall that ***R*$_{3}$** has ~11MB/s read throughput, while ***R*$_{4}$** had ~25MB/s read throughput. Note that ~33MB/s allocation rate equates to 3 * ~11MB/s read throughput. Furthermore, note that ~75MB/s allocation rate equates to 3 * ~25MB/s read throughput. In the next chapter, we'll see where this particular scaling factor of 3 comes from.

Conclusion

To summarize, while reading `data.txt`, Notepad peaked at 1.5GB of memory usage. After reading the file, Notepad gave 500MB back to the system and stabilized at 1GB memory footprint. Memory allocation growth rate during this scenario was roughly three times its read throughput rate. During I/O contention slower read throughput lead to significantly slower memory allocation growth.

It should be clear by now that Notepad is not using memory very efficiently. With 5GB of system memory we were lucky to have enough available memory to accommodate Notepad's hungry memory habits. Any

guesses on what would have happened if our system didn't have enough available memory? When Windows system runs out of available physical memory, it starts relying heavily on the page file located on disk in order to try and supplement physical memory's capacity. When this happens, the resulting paging activity may slow down user experience across the entire system by several orders of magnitude.

***NOTE**: 5,072MB you see in system configuration is the 5*1,024 – 48MB, where 48MB are reserved by the system.*

So why is Notepad charging us such an extensive memory tax (a whole of 300%)? As we saw, after Notepad finished reading the file from disk, it gave the system back 500MB, which was the size of the file. This suggests that Notepad read the contents of the entire file into memory, and then, after it was done processing the data, this memory was released back to the OS. The remaining 1GB of memory usage was exactly twice the file size. Is a factor of two a coincidence?

To better understand what Notepad was actually up to we would need to look at what code was executing while reading `data.txt` into memory. You might think that next step is an introspective code-review of Notepad code. Luckily, with ETW you don't need to go back to the actual source code. As you'll see in the next chapter, we can rely on a technique known as stack-based CPU sampling to get a glimpse into what code Notepad was executing while opening its large file. Perhaps by seeing what it was doing, we can then deduce where this need for extra memory came from.

Key takeaways

To review what we've learned so far:

- Since all activity on the system ultimately gets serviced by one or more of the underlying system resources, it can be very effective to analyze system bottlenecks by reviewing usage of its resources
- Looking at the resource utilization over time, we can often use pattern matching to identify important regions of interest
- Resource contention can be particularly impactful on shared limited resources with multiple concurrent users. As the number of consumers trying to use the same resource grows, so does the probability for contention between them

  It doesn't take many such consumers for typical (client configuration) storage devices, especially for rotational drives. As little as two concurrent users can oftentimes lead to contention on disk slowing down the respective scenarios

  Things are a bit better with concurrent CPU activities, where oftentimes negligible impact is observed as long as enough processors (or cores) are available

- Resource underutilization symptom can be caused by overserialization, which is an act of performing tasks that could have been parallelized in sequence
- In non-memory intensive scenarios, concurrent usage of CPU and/or disk typically far outweigh the performance impact of concurrent usage of memory. In those scenarios whose performance relies more heavily on caching memory content in fast CPU caches (e.g. L2, L3), concurrent usage of memory can also lead to contention, where each user of memory is

fighting for cache space, pushing out cached content of the other process from the fast caches and requiring recurring trips to slower RAM for data accesses
- The impact of the system itself (i.e. system activity, such as indexing, paging, etc.) has to be taken into account when looking at the performance of any given scenario
- WPA offers CPU, disk, and memory graphs to help analyze usage of these respective shared critical system resources
- WPA lets you view resource usage holistically as well as filter to a specific subset of relevant data (e.g. disk usage by a given process, such as `notepad.exe`)
- Stacked line graphs can be a great tool to visualize additive nature of holistic resource usage

In particular, stacked line graph is a good tool to identify resource usage symptoms (e.g. saturation, contention, underutilization, etc.) while using WPA's resource usage views

- For deeper investigations, WPA lets you zoom in on specific temporal selections, in either the current analysis view, or a newly created view
- Tables in detailed graphs enable you to review individual data points behind their graphical counterparts (e.g. individual disk I/Os we saw in the `Disk Usage` graph)
- Contrasting, side by side, two usage graphs for different resources, filtered to the same process, enables us to visually identify cross-resource usage patterns for that process (e.g. overserialization induced by interleaving of disk and CPU usage)

In this chapter, we've looked at system and application activities from the resource usage point of view. While this has enabled us to find where the user scenario has bottlenecked, we don't yet know how to find out why. Looking at what code was executing on the CPU during the regions of interest can oftentimes help us get a better sense of what specific activities lead to the issues at hand.

In the next chapter, we are going to continue making progress on our path towards root cause analysis of Notepad's unresponsiveness, by focusing on the logical activities that took place during the regions of interest that we've identified in this chapter.