

Part 1: Processes

1a. Memory usage of a process

Random Access Memory(RAM) stores temporary memory where the task manager helps to show how much physical and virtual memory is available and how much each program uses it. In figure 1.1, the memory column refers to the percentage of the memory I have used. Under the Memory column, specifies how much memory has been “eaten” by each application. Likewise, Apex Legends uses approximately 3.5GB of memory. The fact is that when games are started, they will read and write data to RAM instead of keeping retrieved from a hard disk and causing inefficiency. Furthermore, google chrome also consume a huge amount of RAM due to chrome splitting every tab into a process. Hence, if a tab crashes, it won't affect the entire application.

Name	Status	35% CPU	71% Memory	2% Disk	1% Network
Apps (4)					
Apex Legends		31.7%	3,584.9 MB	5.9 MB/s	0.6 Mbps
Google Chrome (26)		0%	1,008.4 MB	0 MB/s	0 Mbps
Microsoft Word (2)		0%	90.4 MB	0 MB/s	0 Mbps
Task Manager		0.1%	76.8 MB	0 MB/s	0 Mbps
Background processes (114)					
AcPowerNotification (32 bit)		0%	3.6 MB	0 MB/s	0 Mbps
AMD Crash Defender Service		0%	0.5 MB	0 MB/s	0 Mbps
AMD External Events Client M...		0%	1.5 MB	0 MB/s	0 Mbps
AMD External Events Service ...		0%	0.5 MB	0 MB/s	0 Mbps
Application Frame Host		0%	4.5 MB	0 MB/s	0 Mbps
ARMOURY CRATE (2)		0%	2.5 MB	0 MB/s	0 Mbps
Armoury Crate Control Interfa...		0%	0.7 MB	0 MB/s	0 Mbps
ARMOURY CRATE DenoiseAI		0%	1.0 MB	0 MB/s	0 Mbps
ARMOURY CRATE Service		0%	6.9 MB	0 MB/s	0 Mbps

Figure 1.1

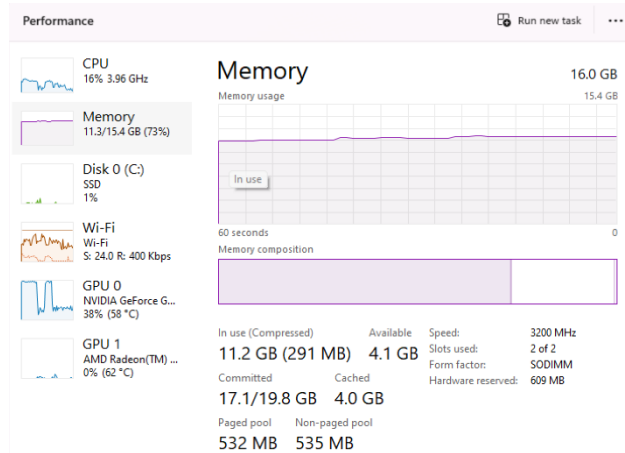


Figure 1.2

1b. CPU usage of a process

On the other hand, the task manager shows the percentage of CPU processes used, the clock speed and each cores usage by dividing the graph into 4x4 graphs. Likewise, each graph represents the corresponding cores. Moreover, the clock speed will increase depending on the workload of the CPU. Thus, the clock speed and the graph will increase if an application uses a lot of CPU. For instance, before “apex legends” run, CPU usage is 6%, the clock speed is 3.45GHz, and the graphs are stable. After “apex legends” run, CPU usage increased to 47%, clock speed increased to 3.81 GHz, and the graph climbed up instantly as there were many calculations needed to be done for the application. According to figure 1.5, 38.1% of CPU has been used by Apex Legends. On the contrary, some processes have less CPU usage, like Desktop Window Manager uses 1.8% of the CPU, and Client Server Runtime Process uses 1.5% of the CPU.

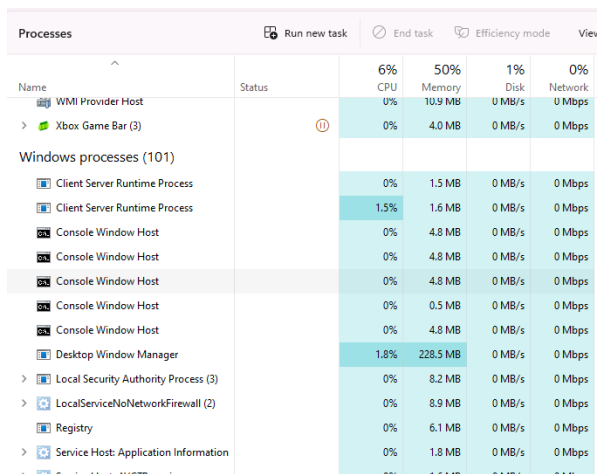


Figure 1.3

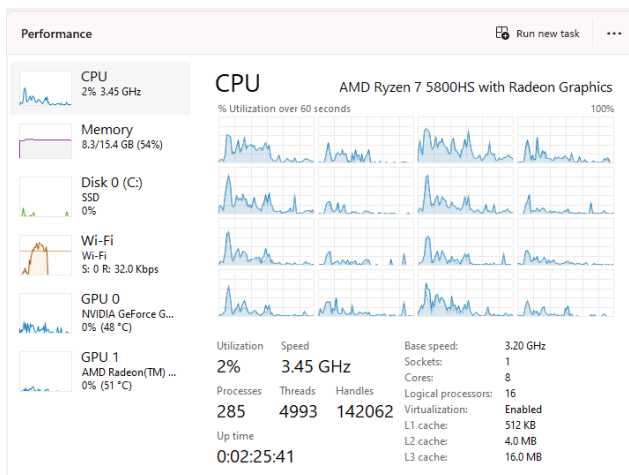


Figure 1.4

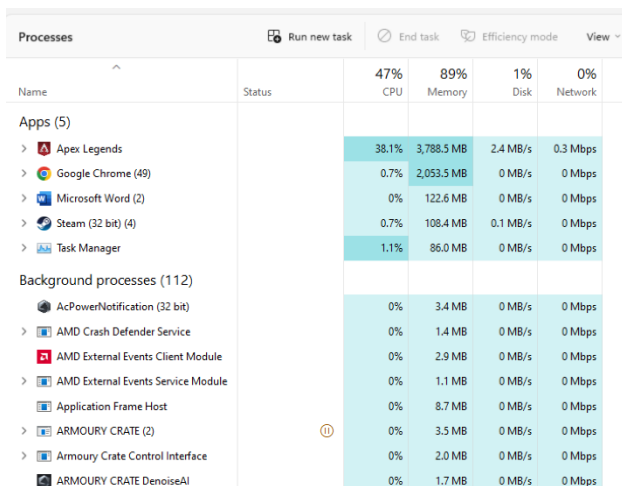


Figure 1.5

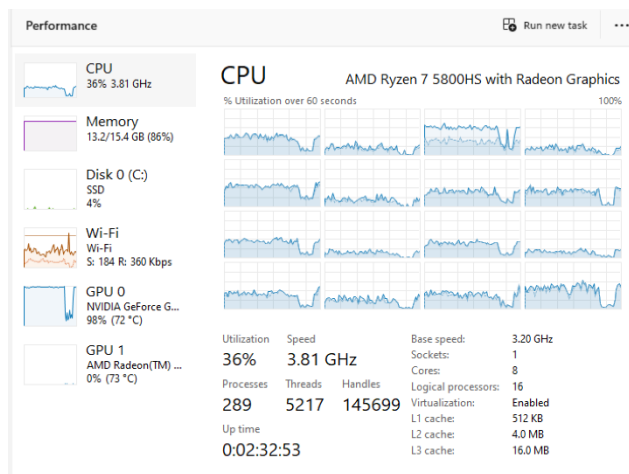


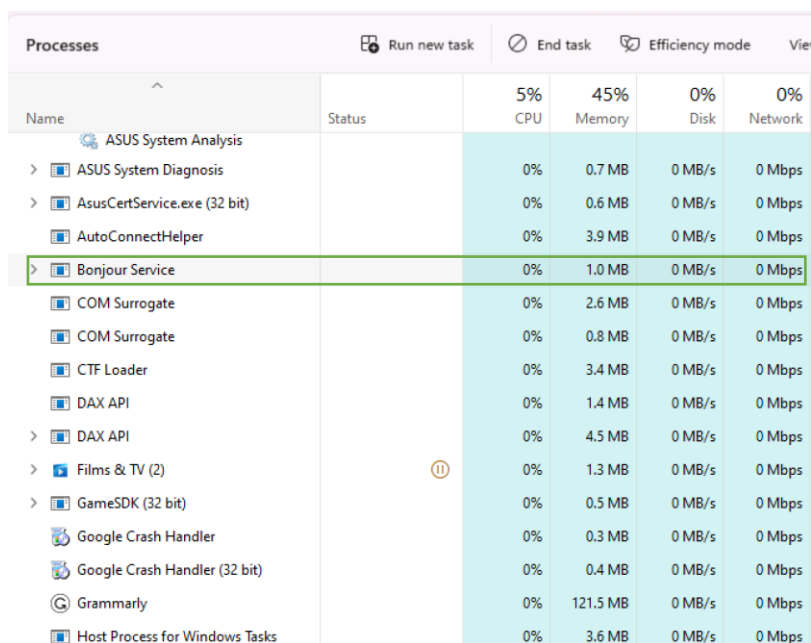
Figure 1.6

Memory usage compared to RAM installed.

The memory available will be less than the RAM installed in the computer. Due to the fact that, the operating system keeps a portion of ram for its usage. Thus, although I have 16GB RAM installed on my computer, only 15.4GB is available. The overall memory usage of all processes in my computer is 11.2GB and the graph of memory in task manager remain stable. Hence, 16GB RAM is sufficient for my usage.

2. Bonjour service is one of the processes that run on my computer. It exists because apple devices can communicate with each other easily, but when it comes to Mac and Windows. It is hard to enable these two systems to share data and services. Thus, Bonjour services are created to enable Apple products to communicate with non-Apple products.

Bonjour service is a small application that silently works in the background, so it doesn't use much internet and disk space. It enables users to set up a local area network(LAN) without configuration. For instance, Bonjour enables Apple and Windows OS to share resources and communicate without configuration settings. With the help of Bonjour service, it enables us to connect to other devices like scanners, printers etc. Bonjour service uses link-local addressing to assign IP addresses instead of Dynamic Host Configuration Protocol (DHCP). It works perfectly fine with IPv4 and IPv6 addressing. Furthermore, apps like iTunes and Safari use Bonjour service to communicate with devices.



Name	Status	5% CPU	45% Memory	0% Disk	0% Network
ASUS System Analysis					
> ASUS System Diagnosis		0%	0.7 MB	0 MB/s	0 Mbps
> AsusCertService.exe (32 bit)		0%	0.6 MB	0 MB/s	0 Mbps
AutoConnectHelper		0%	3.9 MB	0 MB/s	0 Mbps
> Bonjour Service		0%	1.0 MB	0 MB/s	0 Mbps
COM Surrogate		0%	2.6 MB	0 MB/s	0 Mbps
COM Surrogate		0%	0.8 MB	0 MB/s	0 Mbps
CTF Loader		0%	3.4 MB	0 MB/s	0 Mbps
DAX API		0%	1.4 MB	0 MB/s	0 Mbps
> DAX API		0%	4.5 MB	0 MB/s	0 Mbps
> Films & TV (2)		0%	1.3 MB	0 MB/s	0 Mbps
> GameSDK (32 bit)		0%	0.5 MB	0 MB/s	0 Mbps
Google Crash Handler		0%	0.3 MB	0 MB/s	0 Mbps
Google Crash Handler (32 bit)		0%	0.4 MB	0 MB/s	0 Mbps
Grammarly		0%	121.5 MB	0 MB/s	0 Mbps
Host Process for Windows Tasks		0%	3.6 MB	0 MB/s	0 Mbps

Figure 1.7

3. The operating system (OS) must manage the files stored on a computer to protect the users from other processes (eg. buggy code, malicious) accessing our files, which could expose privacy to the public. Thus, the OS act as a middleman between the application and the hardware. Besides that, OS hides complexity for the user and provides a user-friendly interface as they don't need to know what is running in the background and which processes run simultaneously. Thus, this provides a user-friendly interface to the user. Moreover, OS manages the files to enable programmers to access system resources easily.

Part 2: MARIE Programme

Task 2.1: Printing name

Test Case: TehJiaXuan



Figure 2.1.1 Printing Name

Task 2.2: Subroutine prints out string

Test Case: TehJiaXuan



Figure 2.2.1 Printing String

Task 2.3: Subroutine input string from user

Test Case 1 : Hi!!loveFIT1047

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
1F0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
200	0048	0069	0021	0049	006C	006F	0076	0065	0046	0049	0054	0031	0030	0034	0037	0000
210	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Figure 2.3.1 Input String memory

Test Case 2 : HelloWorld!?!?

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
1F0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
200	0048	0065	006C	006C	006F	0057	006F	0072	006C	0064	0021	003F	0021	003F	0000	0000
210	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Figure 2.3.2 Input String memory

Task 2.4: Subroutine convert to lowercase

Test Case 1:

Input String: [A2]MARIE?Program!

Output String: [a2]marie?program!



Figure 2.4.1 Convert lowercase

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
1D0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
1E0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
1F0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
200	005B	0061	0032	005D	006D	0061	0072	0069	0065	003F	0070	0072	006F	0067	0072	0061
210	006D	0021	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
220	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Figure 2.4.2 Convert lowercase memory

Test Case 2:

Input String: FIT1047gg!?!?

Output String: fit1047gg!?!?

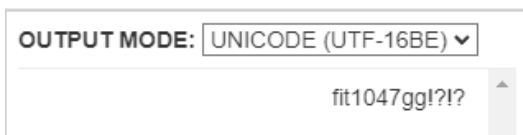


Figure 2.4.3 Convert lowercase

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
1D0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
1E0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
1F0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
200	0066	0069	0074	0031	0030	0034	0037	0067	0067	0021	003F	0021	003F	0000	0000	0000
210	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Figure 2.4.4 Convert lowercase memory

[illegible]

Figure 2.5.6 Memory ROT13 String

Task 2.6: Nested subroutines

Test Case 1:

Input String: [A2]MARIE?Program!!

Output String: [A2]MARIE?Program!!
[a2]marie?program!!
[n2]znevr?cebtenz!!

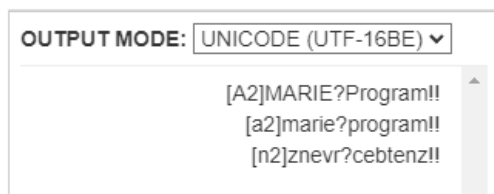


Figure 2.6.1 Convert to encode string.

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
1F0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
200	005B	006E	0032	005D	007A	006E	0065	0076	0072	003F	0063	0065	0062	0074	0065	006E
210	007A	0021	0021	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
220	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Figure 2.6.2 Memory encode string

Test Case 2:

Input String: give-ME-HD!@#?[]

Output String: give-ME-HD!@#?[]
give-me-hd!@#?[]
tvir-zr-uq!@#?[]



Figure 2.6.3 Convert to encode string.

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
1F0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
200	0074	0076	0069	0072	002D	007A	0072	002D	0075	0071	0021	0040	0023	003F	005B	005D
210	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
220	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Figure 2.6.4 Memory encode string

Test case 3:

Input String: tHAnkYou-!VerY-MuchH

Output String: tHAnkYou-!VerY-MuchH
thankyou-!very-much
gunaxlbh-!irel-zhpu

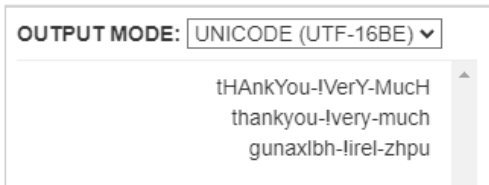


Figure 2.6.5 Convert to encode string.

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
1F0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
200	0067	0075	006E	0061	0078	006C	0062	0068	002D	0021	0069	0072	0065	006C	002D	007A
210	0068	0070	0075	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
220	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Figure 2.6.6 Memory encode string