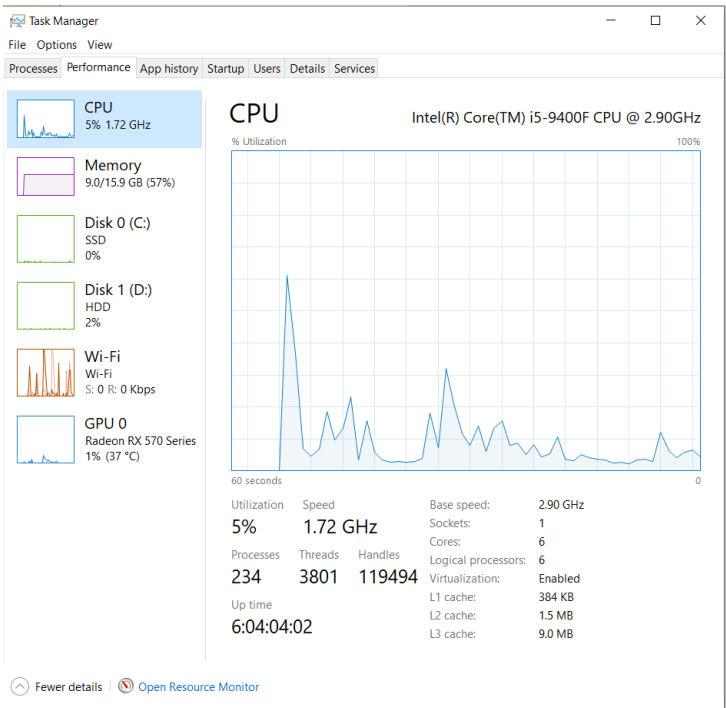
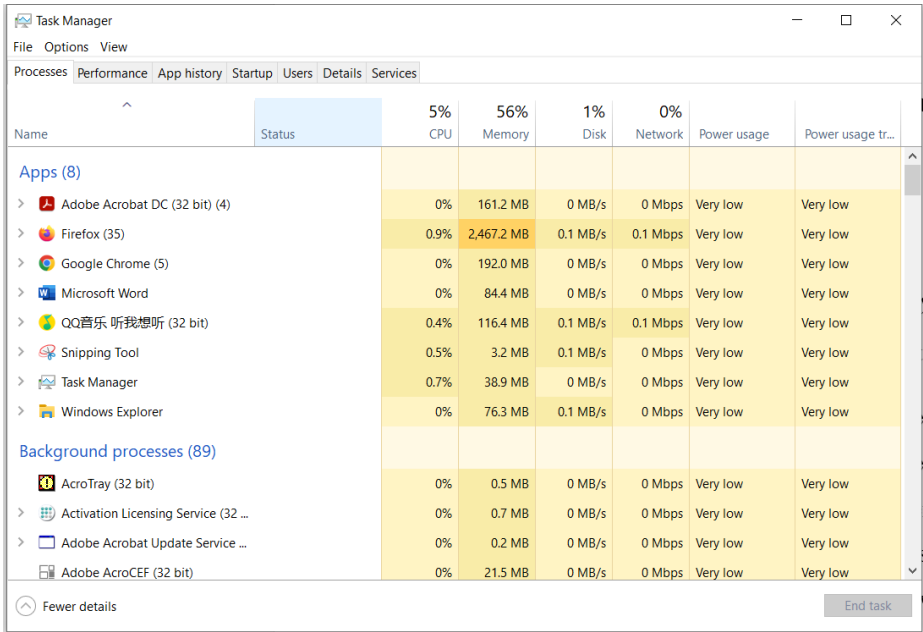


Part 1: Processes (589 words in total)

Task 1.1: (228 words)



In task manager, the first column is the CPU usage and the second is memory usage. The device is powered by an Intel i5 CPU sitting on 16-Gb memory space.

It was first noted that a lot more processes running in the system (background) as compared to the number of opened applications. More interesting, over 56% of the memory (>9 Gb) was



occupied. However, by adding up the memory of all processes in the task manager, the presented usage (~4Gb) is measurably less. This is explained that the task manager is a high-level overviewing platform that only shows the selected processes. Whereas, for example, the windows system itself takes a large portion of the memory while running.

Additionally, for a given process, its CPU usage and memory usage are not related. For example, internet explorer takes more than 2.4 Gb of memory while using only 0.6% of the CPU. This is because many websites are all opened simultaneously - all memory but need little computational power. On the contrary, as typing in Word is happening, the CPU usage spike up to 16% with only 136Mb of memory took – CPU usage with typing but little data being written. Lastly, the CPU usage fluctuates whereas memory usage is relatively stable for a process – when not actively being used it would not require CPU, but its memory space remains on RAM.

Task 1.2: (217 words)

Picked process name: COM surrogate – operated by dllhost.exe

From Task Manager Processes

>  Service Host: Local Service (Net...	0%	1.8 MB	0 MB/s	0 Mbps	Very low	Very low
COM Surrogate	0%	1.8 MB	0 MB/s	0 Mbps	Very low	Very low
 Adobe Collaboration Synchroni...	0%	1.8 MB	0 MB/s	0 Mbps	Very low	Very low

dllhost.exe	5716	Running	SYSTEM	00	888 K	Not allowed
dllhost.exe	5480	Running	RLZ	00	1,872 K	Disabled
dllhost.exe	23148	Running	RLZ	00	856 K	Enabled

From online research on COM Surrogate, it is understood that the abbreviate – COM stands for Computer Object Management. Microsoft originally introduced this back in 1993 to assist/extend other applications with the capability of creating and operating “COM objects”. Accordingly, the COM was developed as a versatile tool that could run in/with different programming environments. One typical example of a COM object operation is the thumbnails in Windows file explorer (file manager). Upon opening a folder which contains media files (e.g., images or videos), the COM object running within the file manager processes these files and creates thumbnail images for display.

One issue with the original COM was that the COM runs within the 'mother' application as a single process. Subsequently, the failure of COM often causes the collapse of the main process. That is, a failed thumbnail generation crashes the entire application. To address this, Microsoft developed the COM Surrogate, which enabled the same functionality of COM object as its original form while allowing it to be run on a separate/independent process. This effectively protects the host application from this side extension. In a modern windows system, the COM surrogate is one essential process for computer operation.

Task 1.3: (144 words)

If each application manages the files itself, users must use different approaches to access the files between different applications. This will cause inefficiency and confusion. Operating system provides a unified way for the users and programmers to store and access data between different applications. Therefore, it allows the user to quickly locate and access the files. Sharing files between different users becomes easy as well.

Another unique function of the operating system is that it uses virtualisation to provide abstractions, which hides the complexity between consistent interfaces to users and programmers. This allows users to implement some functions without understanding the complex principles behind them.

As for security, it protects memory between processes by using virtual memory, which ensures the users' confidential information in one process won't be accessed by another process. It also protects the files of each user on the same computer.

Task 2.3:

The memory addresses store my name starts from 000F.

Assembly code:

```
1 Load PrintString / Load the string address
2 Store subPrintArg / Store in subroutine argument
3 JnS subPrint / Store PC at subPrint, and jump to Loop
4 Halt / Will halt after exit from the subroutine
5
6 subPrintArg, HEX 0 / Space for argument
7 subPrint, HEX 0 / Space for return address
8 Loop, Load subPrintArg / Load from the address stored in the argument
9 Skipcond 800 / Check if the character is greater than 0, if so, skip the next line of code
10 JumpI subPrint / If it's equal to or less than 0, then exit the subroutine(end of string)
11 Output / Output the character
12 Load subPrintArg / Load the PrintFrom address
13 Add One
14 Store subPrintArg / These three lines increment the address by one, which moves to the next character's address
15 Jump Loop / Then jump back to the start of the loop
16
17 PrintString, Adr PrintFrom / Store the address of the "PrintFrom"
18 PrintFrom, HEX 5A / This is the start of the String.
```

Machine halted normally.

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
000	100E	2804	0005	7000	0003	D004	8800	C005	6000	1004	301A	2004	9036	000F	005A	
010	0065	0063	0061	006E	004C	0069	0075	007E	0021	0030	0001	0000	0000	0030	0000	0000
020	0000	0000	0000	0030	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Task 2.4:

The below screenshot is after inputting “FIT1047”.

Assembly code:

```
1 Load Address / Load the memory address 1E
2 Store OutputArg / Store in the subroutine argument
3 JnS Read / Store PC at Read, and jump to Loop
4 Halt / Will halt after exit from the subroutine
5
6 OutputArg, HEX 0 / Space for argument
7 Read, HEX 0 / Space for return address
8 Loop, Input /Take the user input
9 Skipcond 400 /If user inputs zero, then it will skip the next line
10 Jump Flag / Jump to Flag
11 JumpI Read / Exit the subroutine if inputs 0
12 Flag, StoreI OutputArg / Store the user input number into memory address 1E
13 Load OutputArg
14 Add One
15 Store OutputArg /These three lines increment the memory address by one, which moves to the next memory ad
16 Jump Loop / Jump back to the start of the loop
17
18 One, HEX 1 / Use to increment the address
```

Machine halted normally.

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
000	1010	2004	0005	7000	0025	0003	5000	8400	900A	C005	E004	1004	300F	2004	9006	0001
010	001E	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0046	0049
020	0054	0031	0030	0034	0037	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
030	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
040	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
050	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
060	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
070	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
080	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
090	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0A0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0B0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

The below screenshot is after inputting “ZecanLiu~!”.

```

1 Load Address / Load the memory address 1E
2 Store OutputArg / Store in the subroutine argument
3 JnS Read / Store PC at Read, and jump to Loop
4 Halt / Will halt after exit from the subroutine
5
6 OutputArg, HEX 0 / Space for argument
7 Read, HEX 0 / Space for return address
8 Loop, Input /Take the user input
9 Skipcond 400 /If user inputs zero, then it will skip the next line
10 Jump Flag / Jump to Flag
11 JumpI Read / Exit the subroutine if inputs 0
12 Flag, StoreI OutputArg / Store the user input number into memory address 1E
13 Load OutputArg
14 Add One
15 Store OutputArg /These three lines increment the memory address by one, which moves to the next memory address
16 Jump Loop / Jump back to the start of the loop
17
18 One, HEX 1 / Use to increment the address

```

[illegible]

The below screenshot is for converting “5ABUNc5j”.

[illegible]

The screenshot shows the Assembly code editor with the following code:

```

33
34 Org 100
35 leq 9 / space for a particular character
36
37 A 40 / stands for "H", the character before "L"
38 Z 58 / stands for "I", the character after "S"
39 Convert 20 / use to convert the capital letter to corresponding lower case letter
40
41 Address Addr PrintFrom / Store the address of the string
42 PrintFrom 40 / These hex numbers are "H C G P e s" ?
43 WTV 43
44 WTV 47
45 WTV 66
46 HLX 70
47 HLX 72
48 HLX 78
49 HLX 3F
50 WTV 0 / End of string

```

Below the code, it says "Machine halted normally." and a memory dump is visible.

The memory dump shows the following data:

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
000	1021	2004	0003	2005	0003	2004	2010	8400	0005	0005	801E	8010	9019	1010	401F	
010	0008	9019	1010	3020	6000	1004	301E	2004	8006	1010	6000	9015	0002	0010	0040	005e
020	0010	0002	0010	005E	0147	004E	0070	000E	0070	000E	0010	000E	0010	000E	0010	000E
030	0000	0010	0000	0010	0000	0100	0000	0002	0000	0000	0010	0000	0010	0000	0010	0000

Task 2.6:

The below screenshot is for converting “abp?A!”.

The screenshot shows an assembly simulator interface. The assembly code is as follows:

```
1 JmS sub$0013 / Store PC at sub$0013, and jump to Loop
2
3 start, Load addressStart / Load the memory data (after converted string)
4 Skipcond 800 / Check if the memory contains any character
5 Halt / If the memory contains nothing, then halt
6 JmS sub$0013 / Store PC at sub$0013, and jump to Loop
7 Load addressStart
8 Add One
9 Store addressStart / These three line increment the memory address
10 Jump start / Jump to the start
11
12 sub$0013, HEX 0 / Space for return address
13
14 Loop, Input / Take user input
15 Store Temp / Store the character into Temp
16 Skipcond 400 / Check if the character is zero
17 Jump Flag / If not zero, go to Flag
18 Jump sub$0013 / If it's zero, then exit the subroutine
```

The machine halted normally. The memory dump shows the following values:

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
000	0000	0032	0000	0000	0032	0032	0032	0032	0032	0032	0032	0032	0032	0032	0032	0032
010	0000	001F	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	002A	002B	002C	002D
020	0030	0031	0032	0033	0034	0035	0036	0037	0038	0039	003A	003B	003C	003D	003E	003F
030	0040	0041	0042	0043	0044	0045	0046	0047	0048	0049	004A	004B	004C	004D	004E	004F
040	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	005A	005B	005C	005D	005E	005F

The below screenshot is for converting “ABCxyz!~”.

The screenshot shows an assembly simulator interface. The assembly code is as follows:

```
1 JmS sub$0013 / Store PC at sub$0013, and jump to Loop
2
3 start, Load addressStart / Load the memory data (after converted string)
4 Skipcond 800 / Check if the memory contains any character
5 Halt / If the memory contains nothing, then halt
6 JmS sub$0013 / Store PC at sub$0013, and jump to Loop
7 Load addressStart
8 Add One
9 Store addressStart / These three line increment the memory address
10 Jump start / Jump to the start
11
12 sub$0013, HEX 0 / Space for return address
13
14 Loop, Input / Take user input
15 Store Temp / Store the character into Temp
16 Skipcond 400 / Check if the character is zero
17 Jump Flag / If not zero, go to Flag
18 Jump sub$0013 / If it's zero, then exit the subroutine
```

The machine halted normally. The memory dump shows the following values:

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
000	0000	0032	0000	0000	0032	0032	0032	0032	0032	0032	0032	0032	0032	0032	0032	0032
010	0000	001F	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	002A	002B	002C	002D
020	0030	0031	0032	0033	0034	0035	0036	0037	0038	0039	003A	003B	003C	003D	003E	003F
030	0040	0041	0042	0043	0044	0045	0046	0047	0048	0049	004A	004B	004C	004D	004E	004F
040	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	005A	005B	005C	005D	005E	005F

The below screenshot is for converting “QWERzxc”.

The screenshot shows an assembly simulator interface. The assembly code is as follows:

```
1 JmS sub$0013 / Store PC at sub$0013, and jump to Loop
2
3 start, Load addressStart / Load the memory data (after converted string)
4 Skipcond 800 / Check if the memory contains any character
5 Halt / If the memory contains nothing, then halt
6 JmS sub$0013 / Store PC at sub$0013, and jump to Loop
7 Load addressStart
8 Add One
9 Store addressStart / These three line increment the memory address
10 Jump start / Jump to the start
11
12 sub$0013, HEX 0 / Space for return address
13
14 Loop, Input / Take user input
15 Store Temp / Store the character into Temp
16 Skipcond 400 / Check if the character is zero
17 Jump Flag / If not zero, go to Flag
18 Jump sub$0013 / If it's zero, then exit the subroutine
```

The machine halted normally. The memory dump shows the following values:

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
000	0000	0032	0000	0000	0032	0032	0032	0032	0032	0032	0032	0032	0032	0032	0032	0032
010	0000	001F	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	002A	002B	002C	002D
020	0030	0031	0032	0033	0034	0035	0036	0037	0038	0039	003A	003B	003C	003D	003E	003F
030	0040	0041	0042	0043	0044	0045	0046	0047	0048	0049	004A	004B	004C	004D	004E	004F
040	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	005A	005B	005C	005D	005E	005F

Task 2.7:

The below screenshot is for converting “QWEzxc!”.

The below screenshot is for converting “ASD123f”.

[illegible]

The below screenshot is for converting “abp?A!”.

The screenshot shows the Assembly code editor with the following code:

```

1  sub$R0Y13
2
3  start, Load addressStart / Access the data stored in memory
4  Skipcond 800 / Check to see if there is a data stored in memory
5  Halt / Halt when nothing in the memory address
6  Repeat: Store the result
7  Load addressStart
8  Add One
9  Store addressStart / These three lines increment the address
10 Jump start/ Jump to the start to access the next data in memory address
11
12 sub$R0Y13, HEX 0 / Space for return address
13 Loop: Input: / Take the user input
14         Store Temp / Store the data into Temp
15         Skipcond 400 / Check to see if the user input 0
16         Jump LowerCase / Jump to start the convert
17
18
19
20

```

On the right, the Machine halted normally window shows the following registers and values:

Register	Value
IR	0000
PC	0004
IN	0000
OUT	0021

The OUTPUT MODE is set to UNICODE (UTF-16BE). The output window shows the text "Machine halted normally."