Name: Rawan Al Maawali
Lab report # 9: CPU Performance Monitoring
Email Address: 17-0024@student.gutech.edu.om

1. **on page 2: By looking at the last column id (idle time), which system is under heavy load?**

   To know which system has heavy load, we need to look at four factors:

   a. id (idle time): We can see from the illustration given that most of the instances are with idle time of *zero*. In other words, the second system has so much less idle than the first system, which indicates that the second system is executing the CPU more than the first system.

   b. us (CPU utilization): Executing the processes of the second system led to more *US* than the first system. In other words, the CPU utilization in the second system is higher.

   c. in (interrupts): There are exceedingly more interrupts in the second system than the first system.

   d. cs (context switches): System two has a low number of context switches in its cases compared to the system one.

   From the four factors above, we can conclude that the <u>second system</u> is the one under heavy load.

2. **on page 3: A case study analysis (Sustained CPU Utilization).**

   Sustained CPU utilization means the CPU is completely utilized (highest CPU usage).

   To analyze this case study, we need to consider multiple factors:

   a. r (Run Queue): We can notice that the average length of the run queue is about *2* threads. In other words, an average of 2 threads wait for the CPU to execute them but the CPU is busy executing other threads. However, we can see that the last case, in the illustration given, has a run queue length of *zero*, which means the CPU has been able to execute them without leading to starvation. In general, the run queue length, in this case study system, is suitable and does not affect performance negatively. It is important to mention that there is only one run queue for each processor, and usually if the length of the run queue is less than 10 in a processor, then that is appropriate, or if the average length of run queue in most cases is not more than 3, then it is more efficient.

   b. b (Processes Blocked): We can see from the illustration given that there are zero blocked processes in all cases. In other words, these processes are either done with the I/O wait or do not need I/O. This leads to a higher CPU utilization.

   c. in (interrupts): This is the number of interrupts per second. We can notice that the system has a relatively high number of interrupts in most cases. This means that the system is under heavy load.

   d. cs (Context Switches): This is the number of context switches per second. The less context switches there are, the less heavy the system. In this case, the context switches are relatively low. Which means there are less interrupts or switching between processes, which means the CPU is executing the process in large chunks. We can also observe that, in each run, the context switches are incredibly low compared to the interrupts, which means better performance because fewer context switches mean better process utilization, or better CPU performance.

e. us (CPU Utilization) and sy (kernel Utilization): We notice that the sum of the user processes (us) and the kernel/interrupts processes (sy) results in almost 100% in most cases and it is divided almost 70/30 between user and kernal. Which means that the CPU is fully utilized, giving time for both the kernel and the user processes.

f. Id (CPU idle time): Most cases are zero. In other words, the CPU is fully utilized, better performance. However, this can affect computers with small RAM, but in general, the system is working properly.

3. on page 4: Overloaded Scheduler analysis

a. Let us first compare the context switches with the interrupts: we can observe that, in each run, the interrupt is less than the context switch. In other words, the result of difference between the context switches and the interrupts lead CPU overload. To explain, since the interrupt are lower than the context switches, this means between interrupts, there is relatively a high number of context switches for each process, and the more switches there are, the lower the performance and efficiency. This leads to a decrease in the overall performance.

b. us (CPU utilization of user processes): to calculate the total utilization of the user processes, we need to do the following:

Total user processes: 4 + 8 + 9 + 3 + 7 + 2 + 22 + 12 + 6 + 5 + 6 = 84

Then calculate the total utilization in percentage:

Total user processes * (number of runs/100)

84 * 11/100 = 9.24%

Average user utilization = 84/11 = 7.63%

Conclusion:

The average is less than the total, which means the utilization of user processes is appropriate.

c. r (run queue): The average run time length is 2+2+3+4+5+4+6 = 26

The average run time length = 26/11 = 2.4

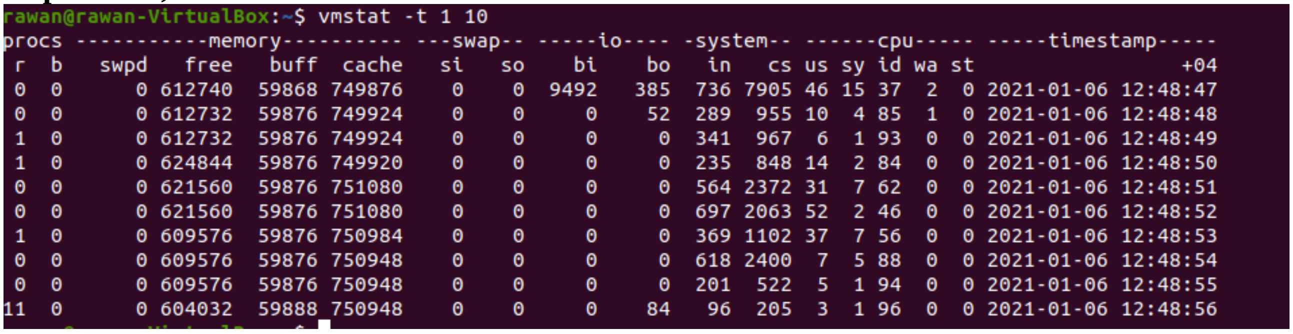
We can notice that the least run time is 0 and the highest is 6. The average 2.4 is between these two values. This means that it is relatively normal. However, for better performance, the length of the run queue is better to be less than 3 in each run and less than 10 threads per run.

Name: Rawan Al Maawali
Lab report # 9: CPU Performance Monitoring
Email Address: 17-0024@student.gutech.edu.om

d. Sy (kernel and interrupt utilization): We can observe that the system kernel processes are taking more of the CPU than the user processes in most runs. In other words, the CPU is overloaded and trying to reschedule priorities.

4. **on page 4&5: Practice section**

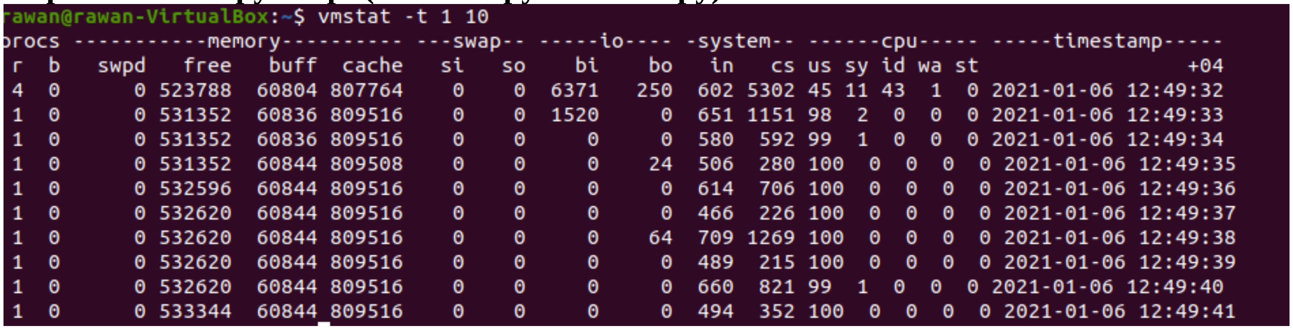
Part 1:
Step1: Run vmstat on your machine with a 1 second delay between updates. Let it run for approximately 10 seconds. Notice the CPU utilization by looking at the (id) column Hint: To stop vmstat, CTRL + C



```
rawan@rawan-VirtualBox:~$ vmstat -t 1 10
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu----- -----timestamp-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st                 +04
 0  0      0 612740  59868 749876    0    0  9492   385  736 7905 46 15 37  2  0 2021-01-06 12:48:47
 0  0      0 612732  59876 749924    0    0     0    52  289  955 10  4 85  1  0 2021-01-06 12:48:48
 1  0      0 612732  59876 749924    0    0     0     0  341  967  6  1 93  0  0 2021-01-06 12:48:49
 1  0      0 624844  59876 749920    0    0     0     0  235  848 14  2 84  0  0 2021-01-06 12:48:50
 0  0      0 621560  59876 751080    0    0     0     0  564 2372 31  7 62  0  0 2021-01-06 12:48:51
 0  0      0 621560  59876 751080    0    0     0     0  697 2063 52  2 46  0  0 2021-01-06 12:48:52
 1  0      0 609576  59876 750984    0    0     0     0  369 1102 37  7 56  0  0 2021-01-06 12:48:53
 0  0      0 609576  59876 750948    0    0     0     0  618 2400  7  5 88  0  0 2021-01-06 12:48:54
 0  0      0 609576  59876 750948    0    0     0     0  201  522  5  1 94  0  0 2021-01-06 12:48:55
11  0      0 604032  59888 750948    0    0     0    84   96  205  3  1 96  0  0 2021-01-06 12:48:56
```

Illustration 1.1: The system vmstat analysis.

Step2: Run ex00.py script (Ubuntu$ python ex00.py). Then run vmstat in another terminal.



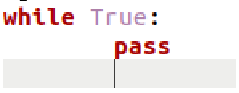
```
rawan@rawan-VirtualBox:~$ vmstat -t 1 10
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu----- -----timestamp-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st                 +04
 4  0      0 523788  60804 807764    0    0  6371   250  602 5302 45 11 43  1  0 2021-01-06 12:49:32
 1  0      0 531352  60836 809516    0    0  1520     0  651 1151 98  2  0  0  0 2021-01-06 12:49:33
 1  0      0 531352  60836 809516    0    0     0     0  580  592 99  1  0  0  0 2021-01-06 12:49:34
 1  0      0 531352  60844 809508    0    0     0    24  506  280 100  0  0  0  0 2021-01-06 12:49:35
 1  0      0 532596  60844 809516    0    0     0     0  614  706 100  0  0  0  0 2021-01-06 12:49:36
 1  0      0 532620  60844 809516    0    0     0     0  466  226 100  0  0  0  0 2021-01-06 12:49:37
 1  0      0 532620  60844 809516    0    0     0    64  709 1269 100  0  0  0  0 2021-01-06 12:49:38
 1  0      0 532620  60844 809516    0    0     0     0  489  215 100  0  0  0  0 2021-01-06 12:49:39
 1  0      0 532620  60844 809516    0    0     0     0  660  821 99  1  0  0  0 2021-01-06 12:49:40
 1  0      0 533344  60844 809516    0    0     0     0  494  352 100  0  0  0  0 2021-01-06 12:49:41
```

Illustration 1.2: The system vmstat analysis after running ex00.py

Q1. What do you notice in this case?
The *r (run time queue)* length increases slightly indicating a process waiting.
The *id (idle time of the CPU)* has dropped to zero in most runs indicating that the CPU – from ex00.py code – pass for one second each run and allowing it to be fully engaged every run because it is executing the code.
The *us (User process utilization)* resembles in this case the code in ex00.py and it engages the CPU fully 100% in most runs, while the kernel processes give up the CPU to allow the code to run.
Q2. Without opening ex00.py, is there anything you can deduce regarding the script
It will be looping infinitely and engaging the CPU for the user process in the script.

Name: Rawan Al Maawali
Lab report # 9: CPU Performance Monitoring
Email Address: 17-0024@student.gutech.edu.om

Q3. Have a look inside ex00.py



```
while True:
        pass
```

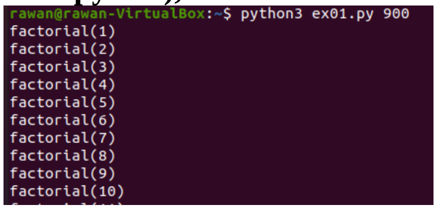
Illustration 1.3: the script inside ex00.py

Q4. Why CPU utilization is around 25% (not 100%)? Hint: run mpstat

In a quad-core we can run 4 threads at once. In our script, we only need one thread, which means we are only using quarter of our entire CPU. This is the reason for most modern computers. However, in our case, the user programs are consuming much more.

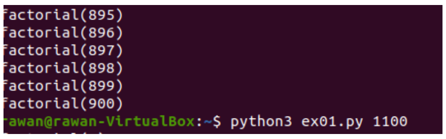
Part2:

Step1: Open ex01.py and run it with a parameter less than 1000 (exp: Ubuntu$ python ex01.py 900), and then more than 1000.



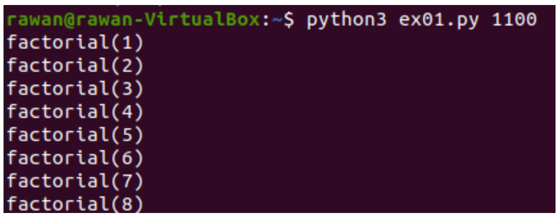
```
rawan@rawan-VirtualBox:~$ python3 ex01.py 900
factorial(1)
factorial(2)
factorial(3)
factorial(4)
factorial(5)
factorial(6)
factorial(7)
factorial(8)
factorial(9)
factorial(10)
```

.
.
.



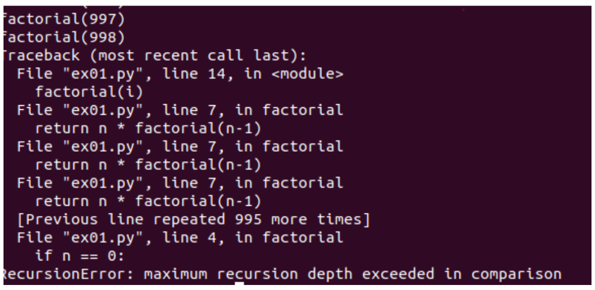
```
actorial(895)
actorial(896)
actorial(897)
actorial(898)
actorial(899)
actorial(900)
awan@rawan-VirtualBox:~$ python3 ex01.py 1100
```

Illustration 2.1: Running ex01.py with a parameter of 900



```
rawan@rawan-VirtualBox:~$ python3 ex01.py 1100
factorial(1)
factorial(2)
factorial(3)
factorial(4)
factorial(5)
factorial(6)
factorial(7)
factorial(8)
```

.
.
.



```
actorial(997)
actorial(998)
raceback (most recent call last):
  File "ex01.py", line 14, in <module>
    factorial(i)
  File "ex01.py", line 7, in factorial
    return n * factorial(n-1)
  File "ex01.py", line 7, in factorial
    return n * factorial(n-1)
  File "ex01.py", line 7, in factorial
    return n * factorial(n-1)
  [Previous line repeated 995 more times]
  File "ex01.py", line 4, in factorial
    if n == 0:
ecursionError: maximum recursion depth exceeded in comparison
```

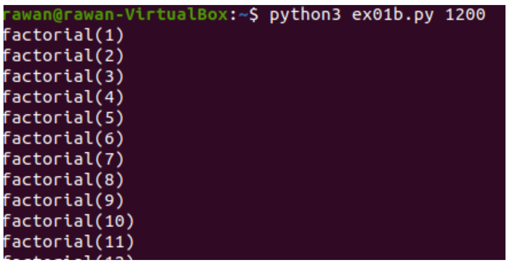
Illustration 2.2: Running ex01.py with a parameter of 1100

Q1. What is the problem?

"RecursionError: maximum recursion depth exceeded in comparison"

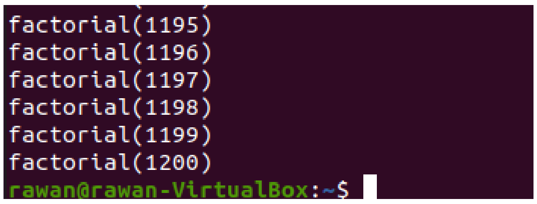
Name: Rawan Al Maawali
Lab report # 9: CPU Performance Monitoring
Email Address: 17-0024@student.gutech.edu.om
Since the script in ex01.py is a recursive function of factorial. The recursion usually stores the current value or state in the system's stack. There is a limit to the system stack, if it is gets full, no more values can be stored. Therefore, factorial(998) got stored in the system stack, but any further recursion will cause the program to fail. When we ran ex01.py with the parameter 1100, we noticed that the program failed. This is because there were too many recursion calls that the system stack got full. There is an overflow.

Step2: Run ex01b.py with a large number (more than 1000). Use vmstat and mpstat to check the CPU behaviour.



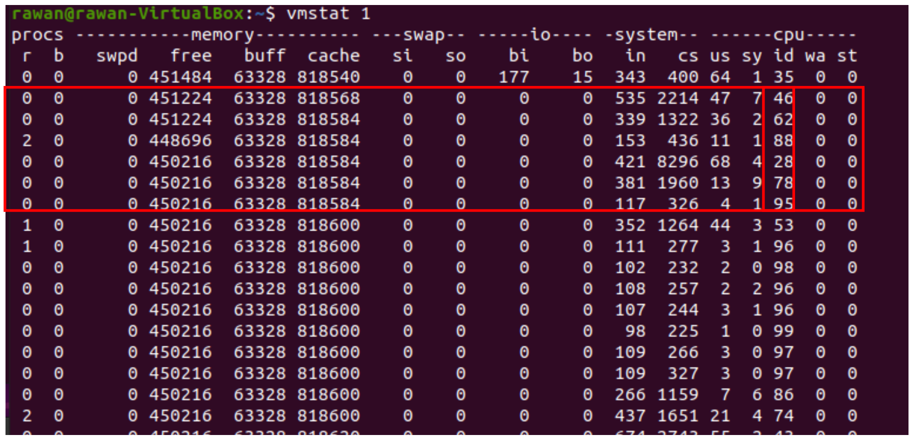
```
rawan@rawan-VirtualBox:~$ python3 ex01b.py 1200
factorial(1)
factorial(2)
factorial(3)
factorial(4)
factorial(5)
factorial(6)
factorial(7)
factorial(8)
factorial(9)
factorial(10)
factorial(11)
```

.
.
.



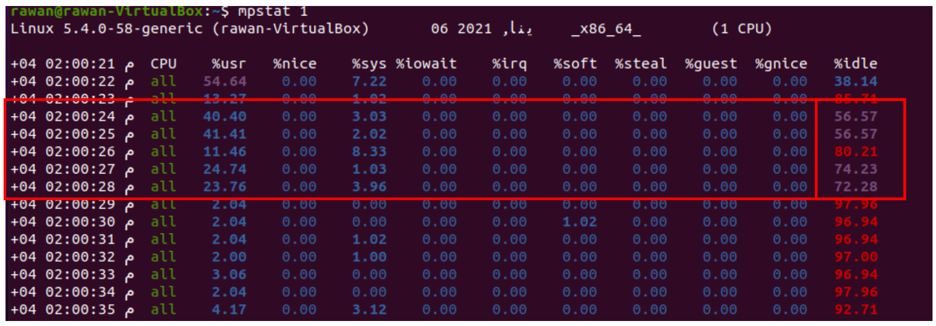
```
factorial(1195)
factorial(1196)
factorial(1197)
factorial(1198)
factorial(1199)
factorial(1200)
rawan@rawan-VirtualBox:~$
```

Illustration 2.3: *Running ex01b.py with a parameter of 1200*



```
rawan@rawan-VirtualBox:~$ vmstat 1
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 451484  63328 818540    0    0   177    15  343  400 64  1 35  0  0
 0  0      0 451224  63328 818568    0    0     0     0  535 2214 47  7 46  0  0
 0  0      0 451224  63328 818584    0    0     0     0  339 1322 36  2 62  0  0
 2  0      0 448696  63328 818584    0    0     0     0  153  436 11  1 88  0  0
 0  0      0 450216  63328 818584    0    0     0     0  421 8296 68  4 28  0  0
 0  0      0 450216  63328 818584    0    0     0     0  381 1960 13  9 78  0  0
 0  0      0 450216  63328 818584    0    0     0     0  117  326  4  1 95  0  0
 1  0      0 450216  63328 818600    0    0     0     0  352 1264 44  3 53  0  0
 1  0      0 450216  63328 818600    0    0     0     0  111  277  3  1 96  0  0
 0  0      0 450216  63328 818600    0    0     0     0  102  232  2  0 98  0  0
 0  0      0 450216  63328 818600    0    0     0     0  108  257  2  2 96  0  0
 0  0      0 450216  63328 818600    0    0     0     0  107  244  3  1 96  0  0
 0  0      0 450216  63328 818600    0    0     0     0   98  225  1  0 99  0  0
 0  0      0 450216  63328 818600    0    0     0     0  109  266  3  0 97  0  0
 0  0      0 450216  63328 818600    0    0     0     0  109  327  3  0 97  0  0
 0  0      0 450216  63328 818600    0    0     0     0  266 1159  7  6 86  0  0
 2  0      0 450216  63328 818600    0    0     0     0  437 1651 21  4 74  0  0
```

Illustration 2.4: vmstat command to check the CPU behavior.



```
rawan@rawan-VirtualBox:~$ mpstat 1
Linux 5.4.0-58-generic (rawan-VirtualBox)       06 2021 ,يناء      _x86_64_        (1 CPU)

+04 02:00:21 م  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
+04 02:00:22 م  all   54.64    0.00    7.22    0.00    0.00    0.00    0.00    0.00    0.00   38.14
+04 02:00:23 م  all   13.27    0.00    1.02    0.00    0.00    0.00    0.00    0.00    0.00   85.71
+04 02:00:24 م  all   40.40    0.00    3.03    0.00    0.00    0.00    0.00    0.00    0.00   56.57
+04 02:00:25 م  all   41.41    0.00    2.02    0.00    0.00    0.00    0.00    0.00    0.00   56.57
+04 02:00:26 م  all   11.46    0.00    8.33    0.00    0.00    0.00    0.00    0.00    0.00   80.21
+04 02:00:27 م  all   24.74    0.00    1.03    0.00    0.00    0.00    0.00    0.00    0.00   74.23
+04 02:00:28 م  all   23.76    0.00    3.96    0.00    0.00    0.00    0.00    0.00    0.00   72.28
+04 02:00:29 م  all    2.04    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   97.96
+04 02:00:30 م  all    2.04    0.00    0.00    0.00    0.00    1.02    0.00    0.00    0.00   96.94
+04 02:00:31 م  all    2.04    0.00    1.02    0.00    0.00    0.00    0.00    0.00    0.00   96.94
+04 02:00:32 م  all    2.00    0.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00   97.00
+04 02:00:33 م  all    3.06    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   96.94
+04 02:00:34 م  all    2.04    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   97.96
+04 02:00:35 م  all    4.17    0.00    3.12    0.00    0.00    0.00    0.00    0.00    0.00   92.71
```

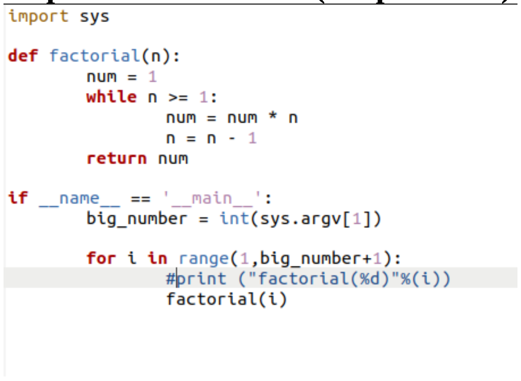
Illustration 2.5: mpstat command to check the CPU behavior.

Name: Rawan Al Maawali
Lab report # 9: CPU Performance Monitoring
Email Address: 17-0024@student.gutech.edu.om

Q2. Explain what is happening.

The script in the ex01b.py contains a function that is iterative and not recursive. Iterative functions do not use the system stack to store values. Iterative functions can take large values of n and they are fast. We notice that we took a parameter of 1200 in this script and the script ran properly without any errors. That is due to the fact the code is an iterative function. We can observe, from illustration 2.4 and 2.5, that the CPU became more engaged during the execution of the code. In other words, the idle time percentage decreased.

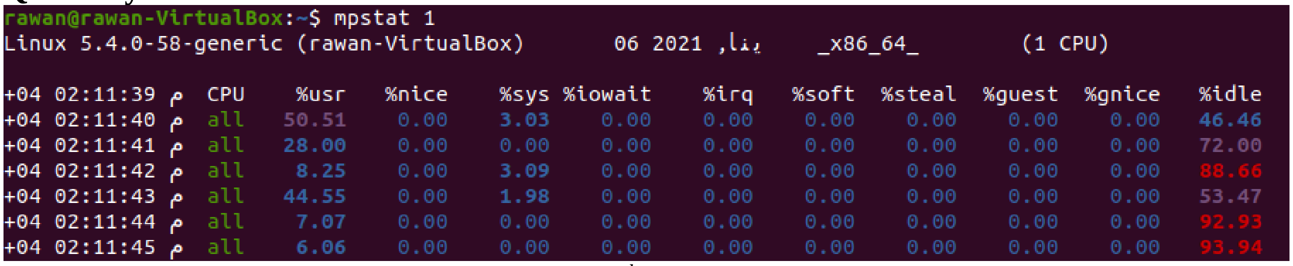
Step3: Remove line 14 (the print call) and then run the program.



```
import sys

def factorial(n):
        num = 1
        while n >= 1:
                num = num * n
                n = n - 1
        return num

if __name__ == '__main__':
        big_number = int(sys.argv[1])

        for i in range(1,big_number+1):
                #print ("factorial(%d)"%(i))
                factorial(i)
```

Illustration 2.6: removing or commenting the 14th line of ex01b.py (the print statement)

Q3. Analyze the behavior now.



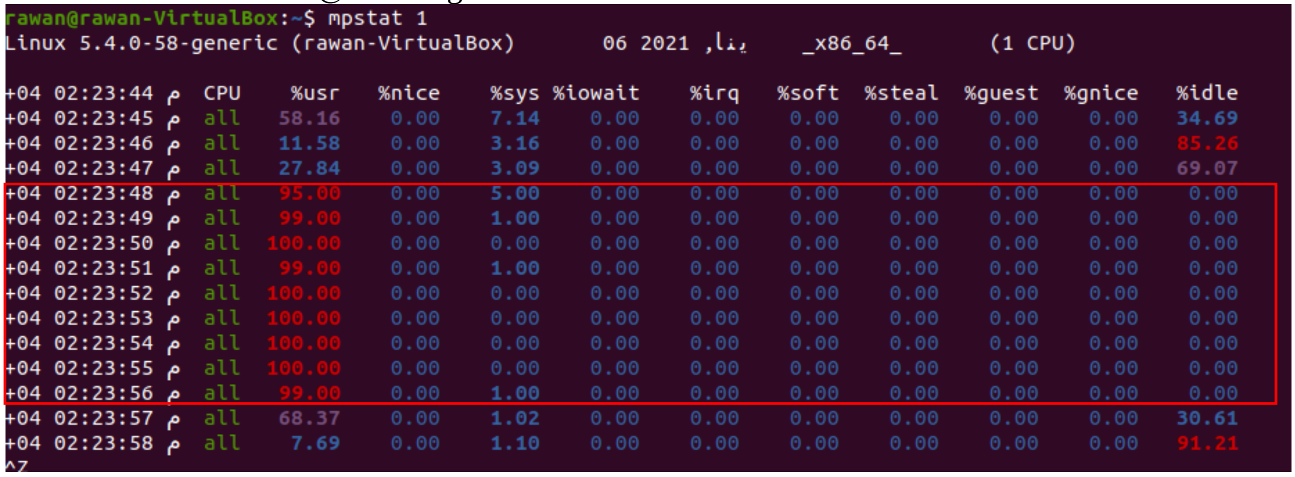
```
rawan@rawan-VirtualBox:~$ mpstat 1
Linux 5.4.0-58-generic (rawan-VirtualBox)       06 2021 ,ﻳﻨﺎ     _x86_64_        (1 CPU)

+04 02:11:39 م  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
+04 02:11:40 م  all   50.51    0.00    3.03    0.00    0.00    0.00    0.00    0.00    0.00   46.46
+04 02:11:41 م  all   28.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   72.00
+04 02:11:42 م  all    8.25    0.00    3.09    0.00    0.00    0.00    0.00    0.00    0.00   88.66
+04 02:11:43 م  all   44.55    0.00    1.98    0.00    0.00    0.00    0.00    0.00    0.00   53.47
+04 02:11:44 م  all    7.07    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   92.93
+04 02:11:45 م  all    6.06    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   93.94
```

Illustration 2.7: removing or commenting the 14th line of ex01b.py (the print statement)

We notice that since we removed the print statement, there will be less instructions for the CPU, therefore more idle time. Comparing mpstat in this case with the previous one (with the print statement), we can see an improvement. We can observe that the CPU has became more idle. Which means less engaged with instructions.

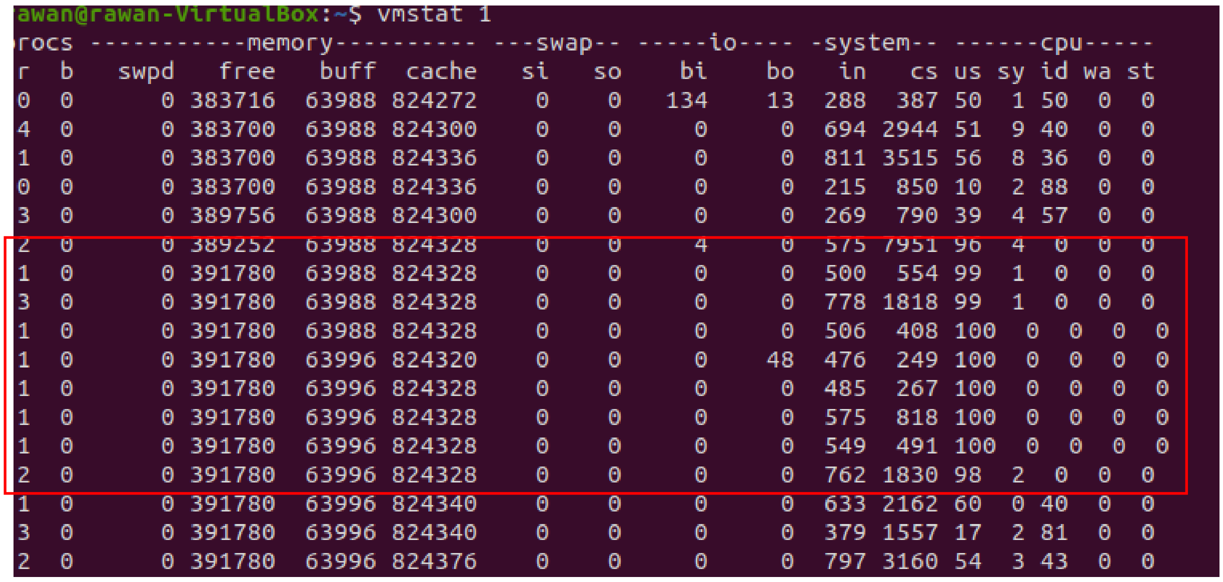
Step4: Run both ex00.py and ex01b.py simultaneously and use vmstat and mpstat to monitor what is going on.

Name: Rawan Al Maawali
Lab report # 9: CPU Performance Monitoring
Email Address: 17-0024@student.gutech.edu.om



```
rawan@rawan-VirtualBox:~$ mpstat 1
Linux 5.4.0-58-generic (rawan-VirtualBox)       06 2021 ,يناير      _x86_64_        (1 CPU)

+04 02:23:44 م  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
+04 02:23:45 م  all   58.16    0.00    7.14    0.00    0.00    0.00    0.00    0.00    0.00   34.69
+04 02:23:46 م  all   11.58    0.00    3.16    0.00    0.00    0.00    0.00    0.00    0.00   85.26
+04 02:23:47 م  all   27.84    0.00    3.09    0.00    0.00    0.00    0.00    0.00    0.00   69.07
+04 02:23:48 م  all   95.00    0.00    5.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
+04 02:23:49 م  all   99.00    0.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
+04 02:23:50 م  all  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
+04 02:23:51 م  all   99.00    0.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
+04 02:23:52 م  all  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
+04 02:23:53 م  all  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
+04 02:23:54 م  all  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
+04 02:23:55 م  all  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
+04 02:23:56 م  all   99.00    0.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
+04 02:23:57 م  all   68.37    0.00    1.02    0.00    0.00    0.00    0.00    0.00    0.00   30.61
+04 02:23:58 م  all    7.69    0.00    1.10    0.00    0.00    0.00    0.00    0.00    0.00   91.21
^Z
```

Illustration 2.8: mpstat command to check the CPU behavior.



```
awan@rawan-VirtualBox:~$ vmstat 1
rocs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
0  0      0 383716  63988 824272    0    0   134    13  288  387 50  1 50  0  0
4  0      0 383700  63988 824300    0    0     0     0  694 2944 51  9 40  0  0
1  0      0 383700  63988 824336    0    0     0     0  811 3515 56  8 36  0  0
0  0      0 383700  63988 824336    0    0     0     0  215  850 10  2 88  0  0
3  0      0 389756  63988 824300    0    0     0     0  269  790 39  4 57  0  0
2  0      0 389252  63988 824328    0    0     4     0  575 7951 96  4  0  0  0
1  0      0 391780  63988 824328    0    0     0     0  500  554 99  1  0  0  0
3  0      0 391780  63988 824328    0    0     0     0  778 1818 99  1  0  0  0
1  0      0 391780  63988 824328    0    0     0     0  506  408 100  0  0  0  0
1  0      0 391780  63996 824320    0    0     0    48  476  249 100  0  0  0  0
1  0      0 391780  63996 824328    0    0     0     0  485  267 100  0  0  0  0
1  0      0 391780  63996 824328    0    0     0     0  575  818 100  0  0  0  0
1  0      0 391780  63996 824328    0    0     0     0  549  491 100  0  0  0  0
2  0      0 391780  63996 824328    0    0     0     0  762 1830 98  2  0  0  0
1  0      0 391780  63996 824340    0    0     0     0  633 2162 60  0 40  0  0
3  0      0 391780  63996 824340    0    0     0     0  379 1557 17  2 81  0  0
2  0      0 391780  63996 824376    0    0     0     0  797 3160 54  3 43  0  0
```

Illustration 2.9: vmstat command to check the CPU behavior.

From illustration 2.8 and 2.9, we can observe that when the two scripts ran together, the CPU was utilized almost completely. Both are taking more CPU. However, after ex01b.py finishes its execution and the infinite loop in ex00.py is still running, this causes a decrease in the CPU utilization.

.