Project completed for CSE 2005

J component

Creation of a Simple Task Manager using Python

UJJWAL GUPTA	**19BLC1133**
SHANE GOMES	**19BCE1081**
VAIBHAV MANGLANI	**19BLC1003**

Abstract

The main theme of our project is to-.

- **Get information about the realtime pocesses running on a computer and the general status of the computer.**

- **In this project we are going to code on Python usingPandas,Psutil etc libraries which will emulate the functionalities of a task manager.**

- **This code will be capable of accessing the kernel and getting real time updates of the processes running on the computer. We are also trying to able to get implicit details such as virtual and swap memory and to find a way of terminating these processes.**

Motivation

- **We wanted to get realtime updates of the processes running in the OS which would help us in knowing how a particular process is identified and how is priority given to a particular process.**
- **Upon searching through the web we found that Psutil library in Python had functionalities that was common to our interest.**
- **Hence we decided to create a simple task manager using Python libraries to get the statistics and status of the processes currently running on the computer.**

Introduction

- **Task manager helps to keep track of the running services(processes,applications etc) Analysis for various research purposes could be performed by the data collected using our program.**

- **We will try to know the current state of the process,the number of cores used ,time of creation etc. We will also try to explore the opportunities to include features to kill/suspend a process if required.**

Objectives

1. **To get real time updates of the processes running on the system.**

2. **Get expliciet information about each process like the read/written bytes required,cores used,the pid and time of creation of the process etc using the Psutil Library.**

3. **To create a dataframe to enlist the processes using Pandas Library.**

4. **To create a record log of the processes in a .CSV file for further analysis.**

Software and Hardware requirements

- A Python IDE –SPYDER in our case.
- Preinstalled Python libraries like Psutil,Pandas,OS etc

Project Explanation

The steps involved in the making of this project are-

- **Psutil Library is used to get real time updates of the os.Pandas is used for dataframe creation.**
- **Process.oneshot- to get process details**
- **Find the time when the process was created .**
- **Find the CPU usage,number of cores that can execute this process:**
- **Find the percentage of CPU resource utilization**
- **Find the the status of the process, whether it is running, sleeping etc**
- **Find the the priority of the process.**
- **Find the Memory Usage of the process**
- **Find the total written and read bytes by a particular process**
- **Find the total number of threads created a process.**
- **Get user info of a process.**
- **Tabulate the information in a dataframe and get out of the loop**
- **Get size() function to convert the size into bytes .**
- **Perform command line argument parsing.**

CODE CREATION:

```
import time
import os
from datetime import datetime
import psutil
import pandas as pd

def proc_information():

    procs = []
    for proc in psutil.process_iter():

        with proc.oneshot():

            proc_identity = proc.pid
            if proc_identity == 0:

                continue

            name = proc.name()
            try:
                name_of_user = proc.username()
            except psutil.AccessDenied:
                name_of_user = "N/A"
            try:
               time_of_creation = datetime.fromtimestamp(proc.create_time())
            except OSError:

               time_of_creation = datetime.fromtimestamp(psutil.boot_time())
            try:

                memory_required = proc.memory_full_info().uss
            except psutil.AccessDenied:
                memory_required = 0

            io_counters = proc.io_counters()
            read_bytes_required = io_counters.read_bytes
            write_bytes_required = io_counters.write_bytes

            total_threads = proc.num_threads()

            try:

                nice_format = int(proc.nice())
            except psutil.AccessDenied:
                nice_format = 0
            try:

                crs = len(proc.cpu_affinity())
```

```
            except psutil.AccessDenied:
                crs = 0

            sys_percent= proc.cpu_percent()

            currstatus = proc.status()




        procs.append({
            'process identity': proc_identity, 'name': name, 'creation_time':t
ime_of_creation,
            'cores': crs, 'cpu_required': sys_percent, 'status': currstatus, '
nice': nice_format,
            'memory_required': memory_required, 'read_bytes_required': read_by
tes_required, 'write_bytes_required': write_bytes_required,
            'total_threads': total_threads, 'username': name_of_user,
        })

    return procs
def size(bytes):

    for unit in ['', 'L', 'Q', 'A', 'G', 'U']:
        if bytes < 1024:
            return f"{bytes:.2f}{unit}B"
        bytes /= 1024

def construct_dataframe(procs):

    df = pd.DataFrame(procs)

    df.set_index('process identity', inplace=True)

    df.sort_values(sort_by, inplace=True, ascending=not descending)

    df['memory_required'] = df['memory_required'].apply(size)
    df['write_bytes_required'] = df['write_bytes_required'].apply(size)
    df['read_bytes_required'] = df['read_bytes_required'].apply(size)

    df['creation_time'] = df['creation_time'].apply(datetime.strftime, args=("
%Y-%m-%d %H:%M:%S",))

    df = df[clmns.split(",")]
    return df

if __name__ == "__main__":
    import argparse
```

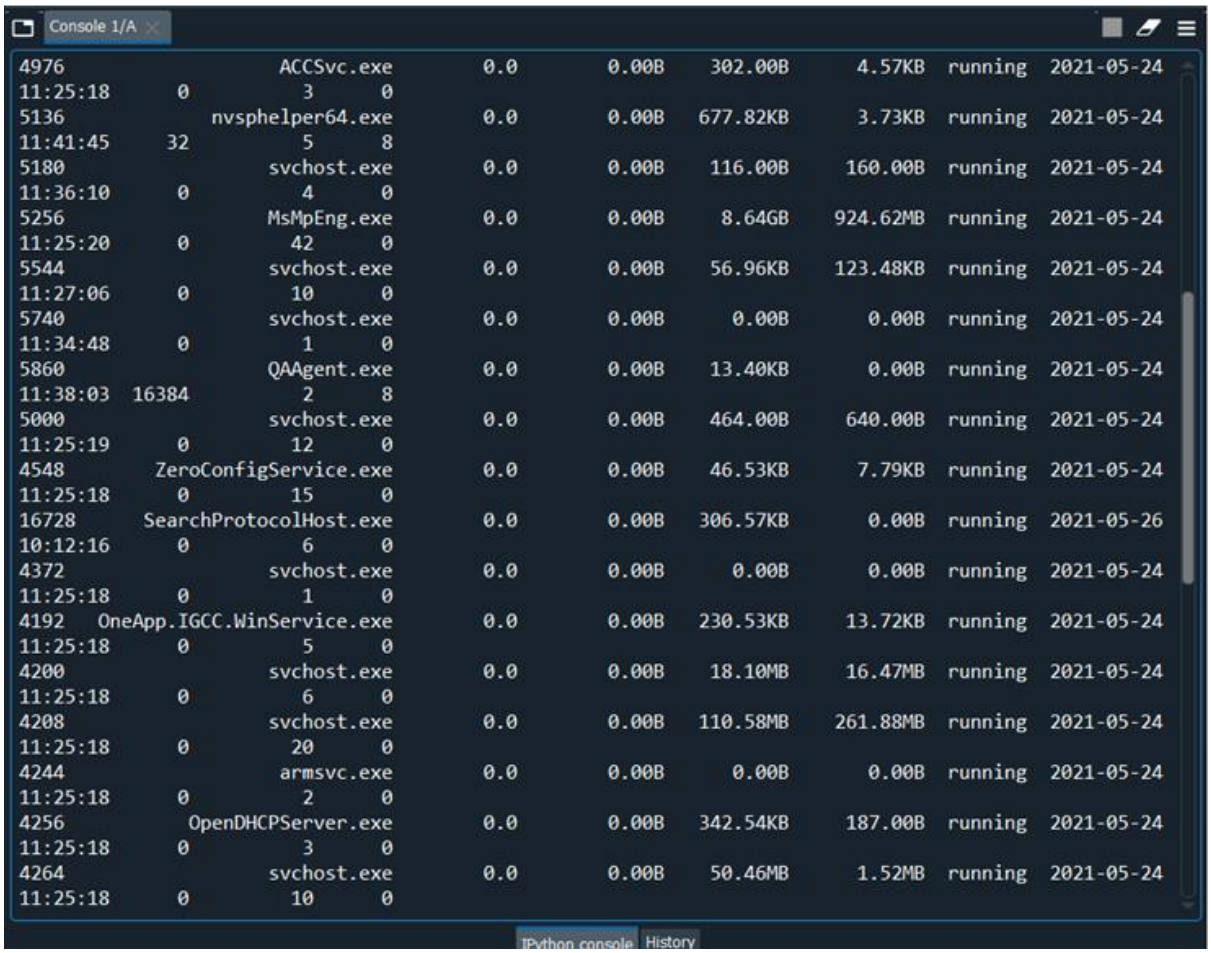
```
    parser = argparse.ArgumentParser(description="proc Viewer & Monitor")
    parser.add_argument("-c", "--
columns", help="""what are the columns to show.

                                                Default is name,cpu_required,m
emory_required,read_bytes_required,write_bytes_required,status,creation_time,n
ice,total_threads,cores.""",
                        default="name,cpu_required,memory_required,read_bytes_
required,write_bytes_required,status,creation_time,nice,total_threads,cores")
    parser.add_argument("-s", "--sort-
by", dest="sort_by", help="Decides from which column shold sorting start.", de
fault="memory_required")
    parser.add_argument("--
descending", action="store_true", help="Decides in which order to sort.")
    parser.add_argument("-
d", help="finds total number of processes,all if 0 is given, default is 20 .",
 default=20)
    parser.add_argument("-u", "--
live_updates_are", action="store_true", help="Decides whether to give live upd
ates")


    args = parser.parse_args()
    clmns = args.columns
    sort_by = args.sort_by
    descending = args.descending
    x = int(args.d)
    live_update = args.live_updates_are

    procs = proc_information()
    df = construct_dataframe(procs)
    if x == 0:
        print(df.to_string())
    elif x > 0:
        print(df.head(x).to_string())

    while live_update:

        procs = proc_information()
        df = construct_dataframe(procs)
        #
        os.system("cls") if "nt" in os.name else os.system("clear")
        if x == 0:
            print(df.to_string())
        elif x > 0:
            print(df.head(x).to_string())
        time.sleep(5)
```

How to use

We can run multiple instances of the program at different points of time and do analysis using the list of the processes in a .CSV File.

Results

process	name	cpu_required	memory_required	read_bytes_required	write_bytes_required	status	creation_time	nice	total_threads	cores	identity
4	System	1.1	0.00B	92.71QB	395.99QB	running	1970-01-01 05:30:00	0	213	0	
4492	SafeExamBrowser.Service.exe	0.0	0.00B	203.41LB	1.03LB	running	2021-06-07 07:28:30	0	4	0	
4528	TiWorker.exe	0.0	0.00B	23.01QB	1.79QB	running	2021-06-07 20:57:53	0	3	0	
4712	svchost.exe	0.0	0.00B	0.00B	0.00B	running	2021-06-07 07:28:30	0	2	0	
4824	armsvc.exe	0.0	0.00B	0.00B	0.00B	running	2021-06-07 07:28:30	0	2	0	
4832	OpenDHCPServer.exe	0.0	0.00B	329.54LB	187.00B	running	2021-06-07 07:28:30	0	3	0	
4912	svchost.exe	0.0	0.00B	0.00B	0.00B	running	2021-06-07 07:28:31	0	1	0	
4968	MsMpEng.exe	1.0	0.00B	3.19AB	644.45QB	running	2021-06-07 07:28:32	0	38	0	
5048	svchost.exe	0.0	0.00B	95.98LB	284.10LB	running	2021-06-07 07:28:33	0	4	0	
5204	svchost.exe	0.0	0.00B	0.00B	0.00B	running	2021-06-07 16:16:46	0	2	0	
5208	svchost.exe	0.0	0.00B	348.00B	480.00B	running	2021-06-07 07:28:34	0	11	0	
5304	QAAdminAgent.exe	0.0	0.00B	282.00B	0.00B	running	2021-06-07 07:33:18	0	2	0	
5344	unsecapp.exe	0.0	0.00B	0.00B	0.00B	running	2021-06-07 07:28:43	0	4	0	
5452	WmiPrvSE.exe	0.0	0.00B	0.00B	0.00B	running	2021-06-07 07:28:43	0	8	0	
5484	rundll32.exe	0.0	0.00B	0.00B	1.42LB	running	2021-06-07 07:28:36	0	2	0	
6100	IAStorDataMgrSvc.exe	0.0	0.00B	530.46LB	9.15LB	running	2021-06-07 07:35:19	0	8	0	
6588	svchost.exe	0.0	0.00B	1.68QB	1.01LB	running	2021-06-08 13:07:02	0	2	0	



SCREENSHOTS FROM .CSV FILE

name	cpu_required	memory_required	read_bytes_required	write_bytes_required	status	creation_time	nice	total_threrads	cores	process identity
System	1.1	0	92.71QB	395.9908	running	01-01-1970 05:30	0	4	0	4492
SafeExamBrowser.Service.exe	0	0	203.41LB	1.03LB	running	07-06-2021 07:28	0	4	0	4528
TiWorker.exe	0	0	23.0108	1.79QB	running	07-06-2021 20:57	0	3	0	4728
svchost.exe	0	0	0	0.00B	running	07-06-2021 07:28	0	2	0	4824
armsvc.exe	0	0	0	0.008	running	07-06-2021 07:28	0	3	0	4832
OpenDHCPServer.exe	0	0	329.54LB	187.008	running	07-06-2021 07:28	0	1	0	4912
svchost.exe	0	0	0	0.008	running	07-06-2021 07:28	0	38	0	4968
MsMpEng.exe	1	0	3.19AB	644.45QB	running	07-06-2021 07:28	0	4	0	5048
svchost.exe	0	0	95.98LB	284.10LB	running	07-06-2021 07:28	0	2	0	5204
svchost.exe	0	0	0.008	0	running	07-06-2021 16:16	0	11	0	5208
svchost.exe	0	0	348.608	480	running	07-06-2021 07:28	0	2	0	5304
QAAdminAgent.exe	0	0	282.008	0	running	07-06-2021 07:33	0	4	0	5344
unsecapp.exe	0	0	0	0	running	07-06-2021 07:28	0	8	0	5452
WmiPrvSE.exe	0	0	0	0	running	07-06-2021 07:28	0	2	0	5484
rund1132.exe	0	0	0	1.42	running	07-06-2021 07:28	0	8	0	6100
IAStorDataMgrSvc.exe	0	0	530.46LB	9.15	running	07-06-2021 07:35	0	2		6588
svchost.exe	0	0	1.6808		running	08-06-2021 13:07	0	4	0	6700
svchost.exe	0	0	2.3508	1.01	running	07-06-2021 07:30	0	4	0	4476

Future Expansion:

In the future we are also trying to make a GUI version of this, with buttons to kill, suspend, and resume the process as there are already available functions for that (`process.kill()`, `process.suspend()` and `process.resume()`).

Conclusion

Thus we were successfully able to implement a command line version of asimple task manager using commonly found Python libraries and display the condumed memory for reading,writing, the number of cores used etc. and record the logs in a .CSV file