

**Department of Computer Science & Engineering**

**NOVEMBER,2021**

**Process Monitor GUI Implementation using Python**

**MINI PROJECT**

**REPORT**

 Student Details :

1. Yash Raj Reg no: 190905056 Roll No: 11 section: B

2. C Sai Kasyap Reg no: 190905084 Roll No: 14 section: B

3. Rishabh Jain Reg no: 190905102 Roll No: 20 section: B

**ABSTRACT**

Process Monitor is essentially used for debugging the processes and systems, effectively managing system resources, and making system decisions. All the running and active real-time processes in the ordered list are displayed.

In this mini project we implemented a graphical user interface for a real-time process monitor where the user will be able to see all the processes, along with their states, CPU usage, memory usage and all the meta-data regarding it.

Users are provided with an option to kill a running process or suspend the process for a while and can resume it whenever required.

Programming Languages used:

The base code for the process monitor is written in python where we have used a python module named psutil which deals with all kinds of tasks such as process resume, suspend, kill and fetch all the meta-data regarding the project.

And then integrated this python code with Tkinter (Python GUI creator) to create the Graphical User Interface for a better visual of the data displayed.

Implementation :

First using Tkinter we have created a window, and then we have created five buttons namely Kill, Suspend, Refresh, Search and Resume.

Then we have also created a search bar where the user can enter the process ID and will be navigated to that section in the list and be shown that process.

Finally we have added a Tkinter Treeview which basically shows all the processes and their data. We have also added a scrollbar so the navigation would become easier.

window = tk.Tk()

window.title('Process Monitor')

window.geometry("1500x900")

searchButton = Button(window,text="Search",command = lambda : searchButtonCallback(search\_entry.get()))

searchButton.grid(row=2,column=0,pady=5,padx=10,sticky='w')

killButton = Button(window,text="Kill", command = lambda : killButtonCallback(int(search\_entry.get())))

killButton.grid(row=3,column=0,pady=5,padx=10,sticky='w')

suspendButton = Button(window, text="Suspend", command = lambda : suspendButtonCallback() )

suspendButton.grid(row=4,column=0,pady=5,padx=10,sticky='w')

resumeButton = Button(window, text = "Resume", command = lambda : resumeButtonCallback() )

resumeButton.grid(row=5, column=0,pady=5,padx=10,sticky='w')

refreshButton = Button(window, text = "Refresh", command = lambda : refreshButtonCallback() )

refreshButton.grid(row=6, column=0,pady=5,padx=10,sticky='w')

style=ttk.Style()

style.theme\_use('clam')

tree = ttk.Treeview(window, columns=columns, show='headings', height=23)

scrollbar = ttk.Scrollbar(window, orient=tk.VERTICAL, command=tree.yview)

tree.configure(yscroll=scrollbar.set)

scrollbar.grid(row=1, column=1, sticky='ns')

After finishing the basic view, we have worked on the logical part of the project i.e. representing the data and linking all the buttons with their corresponding logic.

We used a python module named psutil for fetching the data regarding the processes. We have various methods in the psutil module namely cpu\_percent(), memory\_full\_info(), pid(), name(), username(). All the fetched data is properly formatted and returned to Treeview for properly rendering it.

def data\_generation():

contacts = []

for process in psutil.process\_iter():

# get the process id

with process.oneshot():

pid = process.pid

if pid == 0:

# System Idle Process for Windows NT, useless to see anyways

continue

# get the name of the file executed

name = process.name()

# get the time the process was spawned

try:

create\_time = datetime.fromtimestamp(process.create\_time())

except OSError:

# system processes, using boot time instead

create\_time = datetime.fromtimestamp(psutil.boot\_time())

try:

# get the number of CPU cores that can execute this process

cores = len(process.cpu\_affinity())

except psutil.AccessDenied:

cores = 0

# get the CPU usage percentage

cpu\_usage = process.cpu\_percent()

# get the status of the process (running, idle, etc.)

status = process.status()

try:

# get the process priority (a lower value means a more prioritized process)

nice = int(process.nice())

except psutil.AccessDenied:

nice = 0

try:

# get the memory usage in bytes

memory\_usage = process.memory\_full\_info().uss

except psutil.AccessDenied:

memory\_usage = 0

# total process read and written bytes

memory\_usage = get\_size(memory\_usage)

io\_counters = process.io\_counters()

read\_bytes = io\_counters.read\_bytes

read\_bytes = get\_size(read\_bytes)

write\_bytes = io\_counters.write\_bytes

write\_bytes = get\_size(write\_bytes)

# get the number of total threads spawned by this process

n\_threads = process.num\_threads()

# get the username of user spawned the process

try:

username = process.username()

except psutil.AccessDenied:

username = "N/A"

contacts.append((f'{pid}', f'{name}', f'{cpu\_usage}', f'{memory\_usage}', f'{read\_bytes}', f'{write\_bytes}', f'{status}', f'{create\_time}',f'{nice}', f'{n\_threads}', f'{cores}', f'{username}'))

return contacts

After the data was properly rendered, we worked on the callback functions for the buttons.

The search button callback was written in such a way that when the user enters a PID number in the search bar and hits the search button, the Treeview navigates to that part and highlights that process.

def searchButtonCallback(searchStr):

children = tree.get\_children()

flag=0

for child in children:

if flag==1:

return True

rowData = tree.set(child)

if rowData['pid']==(search\_entry.get().lower()):

tree.selection\_set(child)

tree.see(child)

flag=1

else:

flag=0

if flag==0 :

tk.messagebox.showinfo("SEARCH FAILURE", "<" + searchStr +">" + " was not found in the PID column of the table.")

else:

return True

The kill button callback is written in such a way that when the user enters the PID number in the search bar and hits the kill button, the process is killed and removed from the Treeview.

def killButtonCallback(pid, sig=signal.SIGTERM, include\_parent=True,

timeout=None, on\_terminate=None):

"""Kill a process tree (including grandchildren) with signal

"sig" and return a (gone, still\_alive) tuple.

"on\_terminate", if specified, is a callback function which is

called as soon as a child terminates.

"""

tk.messagebox.showinfo("KILL BUTTON","Kill Button killing : " + search\_entry.get())

if pid == os.getpid():

raise RuntimeError("I refuse to kill myself")

parent = psutil.Process(pid)

children = parent.children(recursive=True)

if include\_parent:

children.append(parent)

for p in children:

p.send\_signal(sig)

gone, alive = psutil.wait\_procs(children, timeout=timeout,callback=on\_terminate)

selection = tree.selection()[0]

tree.delete(selection)

update()

return (gone, alive)

The resume button callback is written in such a way that when the user enters a PID number of any stopped process in the search bar and hits on the resume button, the status of that process is changed from stopped to running.

def resumeButtonCallback():

tk.messagebox.showinfo("RESUME BUTTON","Resume Button resuming : " + search\_entry.get())

some\_pid = int(search\_entry.get())

p = psutil.Process(some\_pid)

print("BEFORE RESUME : " + p.status())

p.resume()

print("AFTER RESUME : " + p.status())

update()

The suspend button callback is written in such a way that when the user enters a PID number of any running process in the search bar and hits on the suspend button, the status of that process is changed from running to stopped.

def suspendButtonCallback():

tk.messagebox.showinfo("SUSPEND BUTTON","Suspend Button suspending : " + search\_entry.get())

some\_pid = int(search\_entry.get() )

p = psutil.Process(some\_pid)

print("BEFORE SUSPEND : " + p.status())

p.suspend()

print("AFTER SUSPEND : " + p.status())

update()

And finally, we have the refresh button callback where the user can click on the button to refresh the data.

def refreshButtonCallback():

tk.messagebox.showinfo("REFRESH BUTTON","Refresh Button refreshing list.")

update()

We have written an update function which will update the data and refresh the Tkinter window.

def update():

updated\_data = data\_generation()

tableData(updated\_data)

window.after(10000,update)

CODE:

**import tkinter as tk**

**import psutil**

**from datetime import datetime**

**from tkinter import ttk**

**from tkinter import \***

**from tkinter import filedialog**

**from tkinter import messagebox**

**import os**

**import signal**

**window = tk.Tk()**

**window.title('Process Monitor')**

**window.geometry("1500x900")**

**"""**

**Function that is called when the search entry box is clicked**

**to remove the placeholder text displayed before.**

**"""**

**def click(\*args):**

**search\_entry.delete(0, 'end')**

**"""**

**Gives the sizes in more readable manner**

**"""**

**def get\_size(bytes):**

**"""**

**Returns size of bytes in a nice format**

**"""**

**for unit in ['', 'K', 'M', 'G', 'T', 'P']:**

**if bytes < 1024:**

**return f"{bytes:.2f}{unit}B"**

**bytes /= 1024**

**"""**

**Searches the treeview table for the string parameter and**

**highlights and jumps to the row it is at, if found.**

**If not found, displays a message box saying not found.**

**"""**

**def searchButtonCallback(searchStr):**

**children = tree.get\_children()**

**flag=0**

**for child in children:**

**if flag==1:**

**return True**

**rowData = tree.set(child)**

**if rowData['pid']==(search\_entry.get().lower()):**

**tree.selection\_set(child)**

**tree.see(child)**

**flag=1**

**else:**

**flag=0**

**if flag==0 :**

**tk.messagebox.showinfo("SEARCH FAILURE", "<" + searchStr +">" + " was not found in the PID column of the table.")**

**else:**

**return True**

**def killButtonCallback(pid, sig=signal.SIGTERM, include\_parent=True,**

**timeout=None, on\_terminate=None):**

**"""Kill a process tree (including grandchildren) with signal**

**"sig" and return a (gone, still\_alive) tuple.**

**"on\_terminate", if specified, is a callabck function which is**

**called as soon as a child terminates.**

**"""**

**tk.messagebox.showinfo("KILL BUTTON","Kill Button killing : " + search\_entry.get())**

**if pid == os.getpid():**

**raise RuntimeError("I refuse to kill myself")**

**parent = psutil.Process(pid)**

**children = parent.children(recursive=True)**

**if include\_parent:**

**children.append(parent)**

**for p in children:**

**p.send\_signal(sig)**

**gone, alive = psutil.wait\_procs(children, timeout=timeout,callback=on\_terminate)**

**selection = tree.selection()[0]**

**tree.delete(selection)**

**update()**

**return (gone, alive)**

**def suspendButtonCallback():**

**tk.messagebox.showinfo("SUSPEND BUTTON","Suspend Button suspending : " + search\_entry.get())**

**some\_pid = int(search\_entry.get() )**

**p = psutil.Process(some\_pid)**

**print("BEFORE SUSPEND : " + p.status())**

**p.suspend()**

**print("AFTER SUSPEND : " + p.status())**

**update()**

**def resumeButtonCallback():**

**tk.messagebox.showinfo("RESUME BUTTON","Resume Button resuming : " + search\_entry.get())**

**some\_pid = int(search\_entry.get() )**

**p = psutil.Process(some\_pid)**

**print("BEFORE RESUME : " + p.status())**

**p.resume()**

**print("AFTER RESUME : " + p.status())**

**update()**

**def refreshButtonCallback():**

**tk.messagebox.showinfo("REFRESH BUTTON","Refresh Button refreshing list.")**

**update()**

**"""**

**Makes the column headings for the treeview**

**"""**

**def createHeadings():**

**tree.column('pid',width=100 ,anchor=CENTER)**

**tree.column('name',width=200 ,anchor=CENTER)**

**tree.column('cpu usage',width=75 ,anchor=CENTER)**

**tree.column('memory usage',width=100 ,anchor=CENTER)**

**tree.column('read bytes',width=100 ,anchor=CENTER)**

**tree.column('written bytes',width=100 ,anchor=CENTER)**

**tree.column('status',width=100 ,anchor=CENTER)**

**tree.column('create time',width=200 ,anchor=CENTER)**

**tree.column('priority',width=50 ,anchor=CENTER)**

**tree.column('thread count',width=100 ,anchor=CENTER)**

**tree.column('cores',width=50 ,anchor=CENTER)**

**tree.column('username',width=150 ,anchor=CENTER)**

**tree.heading('pid', text='PID')**

**tree.heading('name', text='Name')**

**tree.heading('cpu usage', text='CPU Usage')**

**tree.heading('memory usage', text='Memory Usage')**

**tree.heading('read bytes', text='Read Bytes')**

**tree.heading('written bytes', text='Written Bytes')**

**tree.heading('status', text='Status')**

**tree.heading('create time', text='Create Time')**

**tree.heading('priority', text='Priority')**

**tree.heading('thread count', text='Thread Count')**

**tree.heading('cores', text='Cores')**

**tree.heading('username', text='Username')**

**"""**

**Input a list.**

**In the format :**

**('first 1', 'last 1', 'email1@example.com', '1')**

**for however many columns we end up using**

**and in the same order as the column order, so**

**pid, name, cpu, memory, read, written, status, create time, priority, thread count, cores, username**

**"""**

**def tableData(content\_list):**

**for i in tree.get\_children():**

**tree.delete(i)**

**for content in content\_list:**

**tree.insert('', tk.END, values=content)**

**columns = ('pid', 'name', 'cpu usage', 'memory usage', 'read bytes', 'written bytes', 'status', 'create time', 'priority','thread count', 'cores', 'username')**

**style=ttk.Style()**

**style.theme\_use('clam')**

**tree = ttk.Treeview(window, columns=columns, show='headings', height=23)**

**"""**

**Generates data for the UI**

**"""**

**def data\_generation():**

**contacts = []**

**for process in psutil.process\_iter():**

**# get the process id**

**with process.oneshot():**

**pid = process.pid**

**if pid == 0:**

**# System Idle Process for Windows NT, useless to see anyways**

**continue**

**# get the name of the file executed**

**name = process.name()**

**# get the time the process was spawned**

**try:**

**create\_time = datetime.fromtimestamp(process.create\_time())**

**except OSError:**

**# system processes, using boot time instead**

**create\_time = datetime.fromtimestamp(psutil.boot\_time())**

**try:**

**# get the number of CPU cores that can execute this process**

**cores = len(process.cpu\_affinity())**

**except psutil.AccessDenied:**

**cores = 0**

**# get the CPU usage percentage**

**cpu\_usage = process.cpu\_percent()**

**# get the status of the process (running, idle, etc.)**

**status = process.status()**

**try:**

**# get the process priority (a lower value means a more prioritized process)**

**nice = int(process.nice())**

**except psutil.AccessDenied:**

**nice = 0**

**try:**

**# get the memory usage in bytes**

**memory\_usage = process.memory\_full\_info().uss**

**except psutil.AccessDenied:**

**memory\_usage = 0**

**# total process read and written bytes**

**memory\_usage = get\_size(memory\_usage)**

**io\_counters = process.io\_counters()**

**read\_bytes = io\_counters.read\_bytes**

**read\_bytes = get\_size(read\_bytes)**

**write\_bytes = io\_counters.write\_bytes**

**write\_bytes = get\_size(write\_bytes)**

**# get the number of total threads spawned by this process**

**n\_threads = process.num\_threads()**

**# get the username of user spawned the process**

**try:**

**username = process.username()**

**except psutil.AccessDenied:**

**username = "N/A"**

**contacts.append((f'{pid}', f'{name}', f'{cpu\_usage}', f'{memory\_usage}', f'{read\_bytes}', f'{write\_bytes}', f'{status}', f'{create\_time}',f'{nice}', f'{n\_threads}', f'{cores}', f'{username}'))**

**#print(contacts)**

**return contacts**

**"""data gets updated every 10sec by default"""**

**def update():**

**updated\_data=[]**

**updated\_data = data\_generation()**

**tableData(updated\_data)**

**window.after(10000,update)**

**"""**

**Makes the search text box to enter PID.**

**"""**

**search\_entry=Entry(window)**

**search\_entry.grid(row=0,column=0,sticky='w',padx=10)**

**search\_entry.insert(0,"Enter PID to search")**

**search\_entry.bind("<Button-1>", click)**

**tree.grid(row=1, column=0, sticky='nsew',padx=10)**

**"""**

**Adds a scrollbar to the table**

**and assigns it a position on the grid.**

**"""**

**scrollbar = ttk.Scrollbar(window, orient=tk.VERTICAL, command=tree.yview)**

**tree.configure(yscroll=scrollbar.set)**

**scrollbar.grid(row=1, column=1, sticky='ns')**

**"""**

**Makes the search, kill, suspend, resume and refresh buttons**

**and assigns their positions on the grid.**

**"""**

**searchButton = Button(window,text="Search",command = lambda : searchButtonCallback(search\_entry.get()))**

**searchButton.grid(row=2,column=0,pady=5,padx=10,sticky='w')**

**killButton = Button(window,text="Kill", command = lambda : killButtonCallback(int(search\_entry.get())))**

**killButton.grid(row=3,column=0,pady=5,padx=10,sticky='w')**

**suspendButton = Button(window, text="Suspend", command = lambda : suspendButtonCallback() )**

**suspendButton.grid(row=4,column=0,pady=5,padx=10,sticky='w')**

**resumeButton = Button(window, text = "Resume", command = lambda : resumeButtonCallback() )**

**resumeButton.grid(row=5, column=0,pady=5,padx=10,sticky='w')**

**refreshButton = Button(window, text = "Refresh", command = lambda : refreshButtonCallback() )**

**refreshButton.grid(row=6, column=0,pady=5,padx=10,sticky='w')**

**# run the app**

**#update()**

**if \_\_name\_\_=="\_\_main\_\_":**

**"""**

**Creating table headings and inserting data**

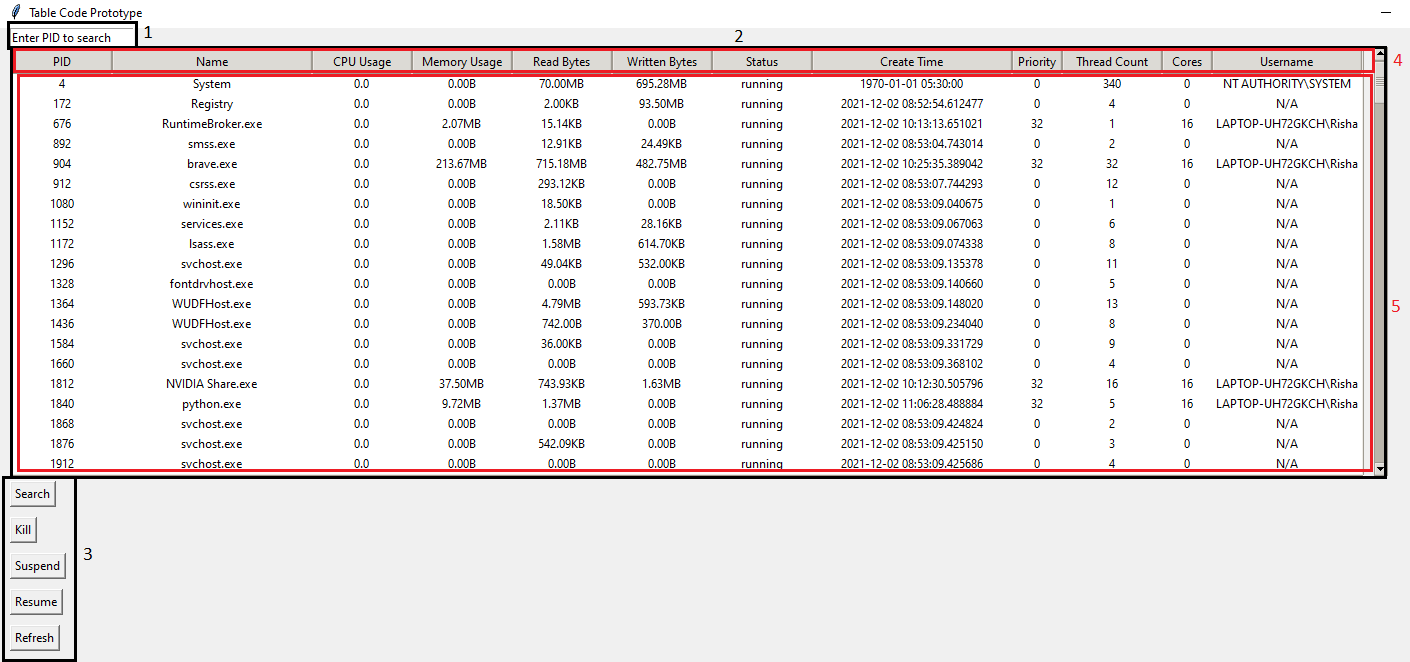
**"""**

**createHeadings()**

**data = data\_generation()**

**tableData(data)**

**window.mainloop()**

UI:

1. Text field for entering the PID for executing actions on it.
2. The table which displays the data of the processes.
3. The buttons (Search, Kill, Suspend, Resume, Refresh) which will be used.
4. The headings of the columns.
5. The actual data obtained through the source code.

Contribution from each student:

C Sai Kasyap worked on data collection of the processes to obtain the values displayed in the GUI and also worked on callback functions which are executed when the respective buttons are pressed.

Yash Raj worked on UI to implement the scrollbar and also worked on data collection of the processes.

Rishabh Jain worked on UI to implement search bar, buttons and treeview and also worked on callback functions.

Learning outcomes:

* To understand all the details regarding a process, process management and memory management.
* To get practical experience as to how the operating system performs process and memory management.
* To analyse the processes and kill or suspend or resume any processes.
* Process monitoring involves a systematic and continuous way of monitoring certain aspects of a program's process which would indicate how well the program is performing.
* Learned about making easy to use and functional GUI, with various button callbacks and functions.

Future scope:

* It can be made lighter, smoother along with being made in a more efficient way.
* It can be used in multiple devices, not only in PCs but also in tablets, mobile devices, IOT etc.
* Creating more user interactive GUI where they can get more details about the processes running in their PCs.
* Creating a function where we can set the limit of memory usage or disk usage and can terminate the process automatically if it crosses the limit.

References:

[1] <https://psutil.readthedocs.io/en/latest/>

[2] <https://docs.python.org/3/library/tkinter.html>

[3] <https://docs.python.org/3/library/os.html>

[4] <https://docs.python.org/3/library/signal.html>