*Problem Statement:*

Memory Consumption and Activity Monitor: Develop an application which is able to identify

the tasks and activities consuming memory resources.

*Source code:*

import psutil

import PySimpleGUI as sg

# Define the layout of the GUI

layout = [

[sg.Text('Process Name'), sg.Text('\tPID'), sg.Text('\tMemory Consumption')],

[sg.Text('\_' \* 80)],

\*[[sg.Text('', size=(20, 1), key=f'process\_name{i}'),

sg.Text('', size=(8, 1), key=f'pid{i}'),

sg.Text('', size=(20, 1), key=f'memory{i}')] for i in range(20)],

[sg.Text('\_' \* 80)],

[sg.Text('CPU Usage:'), sg.Text('', size=(10, 1), key='cpu\_usage')],

[sg.Exit()]

]

# Create the GUI window

window = sg.Window('Memory Consumption and Activity Monitor', layout)

# Start the event loop

while True:

# Read events from the window

event, values = window.read(timeout=100)

# Get the list of running processes

processes = [(p.name(), p.pid, p.memory\_info().rss) for p in psutil.process\_iter()]

# Sort processes by memory usage (descending)

processes.sort(key=lambda x: x[2], reverse=True)

# Update the GUI with the process information

for i in range(20):

if i < len(processes):

window[f'process\_name{i}'].update(processes[i][0])

window[f'pid{i}'].update(processes[i][1])

window[f'memory{i}'].update(f'{processes[i][2] / (1024\*1024):.2f} MB')

else:

window[f'process\_name{i}'].update('')

window[f'pid{i}'].update('')

window[f'memory{i}'].update('')

# Get the CPU usage percentage and update the GUI

cpu\_usage = psutil.cpu\_percent(interval=1)

window['cpu\_usage'].update(f'{cpu\_usage:.2f}%')

# Exit the program if the 'Exit' button is clicked or the window is closed

if event == sg.WINDOW\_CLOSED or event == 'Exit':

break

# Close the GUI window

window.close()

*Explanation:*



This imports two modules: psutil, which provides system information such as CPU usage and memory consumption, and PySimpleGUI, which is a library for creating GUI applications in Python.

A screenshot of a computer

Description automatically generated with medium confidence

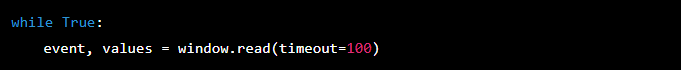
This defines the layout of the GUI using a list of lists. Each nested list represents a row in the GUI. The elements of each row are defined using sg.Text and sg.Exit objects, which display text and provide an exit button, respectively.

The \* operator is used to unpack a list comprehension that generates 20 rows of process information. Each row contains three sg.Text objects that will display the name, PID, and memory consumption of a process.

The key parameter of each sg.Text object is set to a string that includes the index i so that it can be updated later in the program.



This creates a new window with the specified title and layout using the sg.Window method.



This starts an infinite loop that reads events from the window using the window.read method. The timeout parameter is set to 100 milliseconds, which means that the loop will wait for 100 milliseconds before checking for new events.

The event variable contains the name of the event that occurred (e.g., button click), while the values variable contains the current values of any input fields in the GUI.



This creates a list of tuples that contain the name, PID, and memory usage (in bytes) of each running process on the system using the psutil.process\_iter method.



This sorts the list of processes by memory usage in descending order using a lambda function that returns the third element of each tuple (i.e., the memory usage).

Text

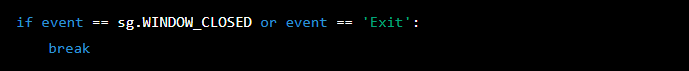
Description automatically generated

This updates the sg.Text objects in the GUI with the process information. It iterates over the first 20 processes in the sorted list and updates the corresponding sg.Text objects with

A screenshot of a computer

Description automatically generated with medium confidence

These lines use the psutil module to get the current CPU usage percentage and update the GUI window with that information. psutil.cpu\_percent() returns the CPU usage as a float value representing the percentage of CPU usage over a certain interval, in this case, 1 second. The value is then formatted as a string with two decimal places using the f-string syntax (f'{cpu\_usage:.2f}%') and updated in the GUI window using the PySimpleGUI method update().



This code block checks if the user has closed the GUI window or clicked on the 'Exit' button, which has a key of 'Exit' in the layout. If either of these events occur, the break statement is executed, which exits the while loop and proceeds to the next line of code.



This line of code closes the PySimpleGUI window and terminates the GUI application. It is executed after the while loop has finished running, either by user interaction (closing the window or clicking the 'Exit' button) or by an exception or error.