**Task manager system**

**المقدمة**

هذا التقرير يوفر شرحاً شاملاً لبرنامج إدارة المهام الذي تم تطويره باستخدام مكتبتي PyQt5 و Psutil في لغة برمجة Python.

البرنامج يقدم واجهة رسومية لمراقبة وإدارة العمليات الجارية على النظام ويوفر معلومات مفصلة حول كل عملية.

**مكتبات البرنامج:**

Psutil:

يُستخدم للوصول إلى معلومات حول العمليات الجارية والاتصالات الشبكية واستهلاك وحدة المعالجة المركزية والذاكرة.

PyQt5:

يُستخدم لتصميم وبناء واجهة المستخدم الرسومية للبرنامج، بما في ذلك الجداول وأزرار الإدارة.

**هيكل البرنامج**

**يتألف البرنامج من الصفوف التالية:**

1. StyledPushButton

* : QPushButton تمثل زرًا في واجهة المستخدم مع تخصيص للمظهر.
* تم تعيين style للزر باستخدام CSS لتحسين المظهر.

1. StyledLabel

* QLabel :تمثل علامة في واجهة المستخدم مع تخصيص للمظهر.
* تم تعيين Style للعلامة باستخدام CSS لتحديد النص بخط عريض.

1. ProcessDetailsWindow:

هذا الصف يُظهر تفاصيل محددة حول عملية نظام تشغيل معينة، حيث يحتوي على التالي:

* + - * قائمة تبويب لعرض الملفات المفتوحة
* المكتبات المحمّلة من قبل العملية.
* زر لحذف ملف محدد.
* تقوم بتحديث عرض الملفات والمكتبات باستخدام دوال

update\_files\_tab &update\_libraries\_tab

1. PerformanceSummaryWidget

* تمثل واجهة لعرض ملخص حول أداء النظام (استخدام وحدة المعالجة المركزية والذاكرة).
* يتم تحديث المعلومات بانتظام باستخدام دالة

update\_performance\_summary.

1. TaskManagerApp:

هذا الصف يمثل النافذة الرئيسية لتطبيق إدارة المهام فيُظهر العمليات الجارية في جدول، حيث يحتوي على التالي:

* جدول يعرض معلومات حول العمليات الجارية، مثل رقم العملية، اسمها، استهلاك الذاكرة، ونسبة استهلاك وحدة المعالجة المركزية . (CPU)
* يحتوي التطبيق على زرين للتنقل بين عرض العمليات وعرض الخدمات، وزر لعرض ملخص الأداء.
* يستخدم QTableWidget لعرض البيانات في شكل جدول، ويتم تحديثه بواسطة الزر "Refresh".
* يُظهر معلومات حول الاتصالات الشبكية لكل عملية.
* يُظهر تفاصيل إضافية حول العملية عند النقر المزدوج عليها.
* يُظهر إدارة العمليات، بما في ذلك حذف ملفات معينة.
* يُظهر ملخص أداء للنظام بشكل عام، مثل استهلاك وحدة المعالجة المركزية واستهلاك الذاكرة.
* يتيح خيار Auto Refresh لتحديث البيانات بانتظام بشكل تلقائي
* يتضمن العديد من الأزرار للتفاعل مع النظام، مثل إنهاء العمليات أو إيقاف الخدمات.
* يتيح التبديل بين عرض العمليات وعرض الخدمات.
* يقوم بتحديث البيانات بانتظام عبر دوال show\_processes و show\_services

**التعليمات الأساسية:**

1. إنشاء التطبيق وعرض النافذة:

app = QApplication(sys.argv)

window = TaskManagerApp()

window.show()

sys.exit(app.exec\_())

1. تحديث الجدول:

* يتم تحديث البيانات في الجدول باستخدام دوال

show\_processes و show\_services و refresh\_table

1. التفاعل مع النظام:

* يتيح البرنامج إنهاء العمليات أو إيقاف الخدمات باستخدام دالة end\_task\_or\_service.
* يتم استخدام مكتبة psutil للتفاعل مع العمليات والخدمات.

1. التحديث التلقائي:

* يوفر خيار Auto Refresh لتحديث البيانات بشكل تلقائي باستخدام دالة

toggle\_auto\_refresh و auto\_refresh

1. عرض تفاصيل العمليات:

* يمكن عرض تفاصيل العمليات عبر دالة show\_process\_details عند النقر المزدوج على صف في الجدول.

1. إنهاء التطبيق:

* يتم إيقاف التطبيق باستخدام sys.exit(app.exec\_()).

**الواجهة الرسومية:**

* تم استخدام PyQt5 لإنشاء واجهة المستخدم، وتم تخصيص الأزرار باستخدام StyledPushButton لتحسين المظهر.
* تم تنظيم العناصر باستخدام QVBoxLayout و QHBoxLayout لتنظيمها بشكل مناسب.

**تحديث البيانات:**

* يتم تحديث البيانات بانتظام باستخدام دالة refresh\_table عند النقر على زر "Refresh" أو بشكل تلقائي إذا تم تحديد خيار "Auto Refresh".
* يتم استخدام psutil للوصول إلى معلومات النظام مثل العمليات والخدمات والأداء.

**التفاعل مع النظام:**

* يوفر التطبيق واجهة لإنهاء العمليات أو إيقاف الخدمات المحددة.

**الأخطاء والاستثناءات:**

* تم التعامل مع الأخطاء والاستثناءات المحتملة باستخدام تصريحات try/except لتجنب توقف التطبيق في حالة حدوث خطأ.

**الحدث الرئيسي:**

* في النهاية، يتم إنشاء تطبيق QApplication وعرض نافذة TaskManagerApp.

**الكود**

**import sys**

**import psutil**

**from PyQt5.QtWidgets import \***

**from PyQt5.QtGui import QIntValidator**

**from PyQt5.QtCore import \***

**import os**

**import subprocess**

**class StyledPushButton(QPushButton):**

**def \_\_init\_\_(self, text):**

**super().\_\_init\_\_(text)**

**self.setStyleSheet("""**

**QPushButton {**

**background-color: #f8f8f8;**

**border: 1px solid #d8d8d8;**

**padding: 5px;**

**margin: 5px;**

**}**

**QPushButton:hover {**

**background-color: #e8e8e8;**

**}**

**""")**

**class StyledLabel(QLabel):**

**def \_\_init\_\_(self, text):**

**super().\_\_init\_\_(text)**

**self.setStyleSheet("font-weight: bold;")**

**class ProcessDetailsWindow(QWidget):**

**def \_\_init\_\_(self, pid):**

**super().\_\_init\_\_()**

**self.setWindowTitle(f"Process Details - PID: {pid}")**

**self.setGeometry(200, 200, 800, 600)**

**self.tab\_widget = QTabWidget()**

**# Tab for files**

**file\_tab = QWidget()**

**file\_layout = QVBoxLayout()**

**self.file\_list\_widget = QListWidget()**

**file\_layout.addWidget(self.file\_list\_widget)**

**delete\_button = StyledPushButton("Delete Selected File")**

**delete\_button.clicked.connect(self.delete\_selected\_file)**

**file\_layout.addWidget(delete\_button)**

**file\_tab.setLayout(file\_layout)**

**# Tab for libraries**

**lib\_tab = QWidget()**

**lib\_layout = QVBoxLayout()**

**self.lib\_list\_widget = QListWidget()**

**lib\_layout.addWidget(self.lib\_list\_widget)**

**lib\_tab.setLayout(lib\_layout)**

**# Add tabs to the tab widget**

**self.tab\_widget.addTab(file\_tab, "Files")**

**self.tab\_widget.addTab(lib\_tab, "Libraries")**

**main\_layout = QVBoxLayout()**

**main\_layout.addWidget(self.tab\_widget)**

**self.setLayout(main\_layout)**

**self.pid = pid**

**# Load initial data**

**self.update\_files\_tab()**

**self.update\_libraries\_tab()**

**def update\_files\_tab(self):**

**self.file\_list\_widget.clear()**

**try:**

**files = [f.path for f in psutil.Process(self.pid).open\_files()]**

**for file in files:**

**self.file\_list\_widget.addItem(file)**

**except Exception as e:**

**print(f"An error occurred while updating files tab: {e}")**

**def update\_libraries\_tab(self):**

**self.lib\_list\_widget.clear()**

**try:**

**libs = [lib for lib in psutil.Process(self.pid).memory\_maps()]**

**for lib in libs:**

**self.lib\_list\_widget.addItem(lib.path)**

**except psutil.AccessDenied:**

**print(f"Access denied. Unable to retrieve information for PID {self.pid}.")**

**except Exception as e:**

**print(f"An error occurred while updating libraries tab: {e}")**

**def delete\_selected\_file(self):**

**selected\_item = self.file\_list\_widget.currentItem()**

**if selected\_item:**

**file\_path = selected\_item.text()**

**print(f"Deleting file: {file\_path}")**

**self.update\_files\_tab()**

**class PerformanceSummaryWidget(QWidget):**

**def \_\_init\_\_(self):**

**super().\_\_init\_\_()**

**self.cpu\_label = StyledLabel("CPU Usage:")**

**self.cpu\_usage\_label = QLabel()**

**self.memory\_label = StyledLabel("Memory Usage:")**

**self.memory\_usage\_label = QLabel()**

**main\_layout = QVBoxLayout()**

**main\_layout.addWidget(self.cpu\_label)**

**main\_layout.addWidget(self.cpu\_usage\_label)**

**main\_layout.addWidget(self.memory\_label)**

**main\_layout.addWidget(self.memory\_usage\_label)**

**self.setLayout(main\_layout)**

**self.update\_performance\_summary()**

**def update\_performance\_summary(self):**

**cpu\_percent\_list = psutil.cpu\_percent(percpu=True)**

**avg\_cpu\_percent = sum(cpu\_percent\_list) / len(cpu\_percent\_list)**

**memory\_info = psutil.virtual\_memory()**

**self.cpu\_usage\_label.setText(f"{avg\_cpu\_percent:.2f}%")**

**self.memory\_usage\_label.setText(f"Used: {memory\_info.used / (1024 \* 1024):.2f} MB, Total: {memory\_info.total / (1024 \* 1024):.2f} MB")**

**class TaskManagerApp(QWidget):**

**def \_\_init\_\_(self):**

**super().\_\_init\_\_()**

**self.setWindowTitle("Task Manager")**

**self.setGeometry(100, 100, 1000, 600)**

**self.setStyleSheet("background-color: white;")**

**self.processes\_button = StyledPushButton("Processes")**

**self.services\_button = StyledPushButton("Services")**

**#self.performance\_summary\_button = StyledPushButton("Performance Summary")**

**self.processes\_button.clicked.connect(self.show\_processes)**

**self.services\_button.clicked.connect(self.show\_services)**

**# self.performance\_summary\_button.clicked.connect(self.show\_performance\_summary)**

**self.table\_widget = QTableWidget()**

**self.table\_widget.setColumnCount(8)**

**self.table\_widget.setHorizontalHeaderLabels(["PID/Service Name", "Name/Display Name", "Memory/Status", "CPU/Start Type",**

**"Local IP", "Local Port", "Remote IP", "Remote Port"])**

**self.end\_button = StyledPushButton("End Task/Stop Service")**

**self.end\_button.clicked.connect(self.end\_task\_or\_service)**

**self.performance\_summary\_widget = PerformanceSummaryWidget()**

**self.layout = QVBoxLayout()**

**button\_layout = QHBoxLayout()**

**button\_layout.addWidget(self.processes\_button)**

**button\_layout.addWidget(self.services\_button)**

**#button\_layout.addWidget(self.performance\_summary\_button)**

**self.layout.addLayout(button\_layout)**

**self.layout.addWidget(self.performance\_summary\_widget)**

**self.layout.addWidget(self.table\_widget)**

**refresh\_layout = QHBoxLayout()**

**self.auto\_refresh\_checkbox = QCheckBox("Auto Refresh")**

**self.auto\_refresh\_checkbox.stateChanged.connect(self.toggle\_auto\_refresh)**

**self.refresh\_interval\_input = QLineEdit(self)**

**self.refresh\_interval\_input.setPlaceholderText("Refresh Interval (seconds)")**

**self.refresh\_interval\_input.setValidator(QIntValidator()) # Only allow integer input**

**refresh\_layout.addWidget(self.auto\_refresh\_checkbox)**

**refresh\_layout.addWidget(self.refresh\_interval\_input)**

**self.refresh\_timer = QTimer(self)**

**self.refresh\_timer.timeout.connect(self.auto\_refresh)**

**self.refresh\_button = StyledPushButton("Refresh")**

**self.refresh\_button.clicked.connect(self.refresh\_table)**

**self.refresh\_button.setFixedSize(80, 40)**

**refresh\_layout.addWidget(self.refresh\_button)**

**refresh\_layout.addStretch() # Add stretchable space to push buttons to the right**

**refresh\_layout.addWidget(self.end\_button)**

**self.layout.addLayout(refresh\_layout)**

**self.setLayout(self.layout)**

**self.details\_windows = {}**

**self.show\_processes()**

**self.show\_processes()**

**self.table\_widget.itemDoubleClicked.connect(self.show\_process\_details)**

**for i in range(self.table\_widget.columnCount()):**

**self.table\_widget.setSortingEnabled(True)**

**self.table\_widget.sortByColumn(i, Qt.AscendingOrder)**

**def toggle\_auto\_refresh(self, state):**

**if state == Qt.Checked:**

**# Start the timer when the checkbox is checked**

**refresh\_interval = int(self.refresh\_interval\_input.text()) if self.refresh\_interval\_input.text() else 1**

**self.refresh\_timer.start(refresh\_interval \* 1000) # Convert seconds to milliseconds**

**else:**

**# Stop the timer when the checkbox is unchecked**

**self.refresh\_timer.stop()**

**def auto\_refresh(self):**

**if a == 1:**

**self.show\_processes()**

**self.show\_performance\_summary()**

**elif a == 2:**

**self.show\_services()**

**self.show\_performance\_summary()**

**def refresh\_table(self):**

**if a == 1:**

**self.show\_processes()**

**self.show\_performance\_summary()**

**#self.update\_performance\_summary()**

**elif a == 2:**

**self.show\_services()**

**self.show\_performance\_summary()**

**def show\_processes(self):**

**global a**

**a = 1**

**self.table\_widget.setColumnCount(8)**

**self.table\_widget.setHorizontalHeaderLabels(["PID", "Name", "Memory", "CPU",**

**"Local IP", "Local Port", "Remote IP", "Remote Port"])**

**self.table\_widget.setRowCount(0)**

**try:**

**for process in psutil.process\_iter(['pid', 'name', 'memory\_info', 'connections', 'cpu\_percent']):**

**pid = process.info['pid']**

**cpu\_percent = process.info['cpu\_percent']**

**process\_name = process.info['name']**

**memory = process.info['memory\_info'].rss / (1024 \* 1024)**

**#cpu\_percent = process.cpu\_percent(interval = 0.01) / psutil.cpu\_count()**

**#cpu\_percent = process('cpu\_percent')**

**row\_position = self.table\_widget.rowCount()**

**self.table\_widget.insertRow(row\_position)**

**self.table\_widget.setItem(row\_position, 0, QTableWidgetItem(str(pid)))**

**self.table\_widget.setItem(row\_position, 1, QTableWidgetItem(process\_name))**

**self.table\_widget.setItem(row\_position, 2, QTableWidgetItem(f"{memory:.2f} MB"))**

**self.table\_widget.setItem(row\_position, 3, QTableWidgetItem(f"{cpu\_percent:.2f}%"))**

**connections = process.info.get('connections', [])**

**for connection in connections:**

**if connection.laddr and connection.raddr:**

**local\_ip, local\_port = connection.laddr**

**remote\_ip, remote\_port = connection.raddr**

**self.table\_widget.setItem(row\_position, 4, QTableWidgetItem(local\_ip))**

**self.table\_widget.setItem(row\_position, 5, QTableWidgetItem(str(local\_port)))**

**self.table\_widget.setItem(row\_position, 6, QTableWidgetItem(remote\_ip))**

**self.table\_widget.setItem(row\_position, 7, QTableWidgetItem(str(remote\_port)))**

**except Exception as e:**

**print(f"An error occurred while updating the processes table: {e}")**

**self.table\_widget.itemDoubleClicked.connect(self.show\_process\_details)**

**def show\_process\_details(self, item):**

**pid = int(self.table\_widget.item(item.row(), 0).text())**

**try:**

**if pid in self.details\_windows:**

**details\_window = self.details\_windows[pid]**

**details\_window.show()**

**else:**

**details\_window = ProcessDetailsWindow(pid)**

**self.details\_windows[pid] = details\_window**

**details\_window.destroyed.connect(lambda: self.details\_windows.pop(pid, None))**

**details\_window.show()**

**except Exception as e:**

**print(f"An error occurred while showing process details: {e}")**

**def end\_task\_or\_service(self):**

**selected\_item = self.table\_widget.currentItem()**

**if selected\_item:**

**identifier = selected\_item.text().split()[0]**

**try:**

**if identifier.isdigit():**

**pid = int(identifier)**

**psutil.Process(pid).terminate()**

**print(f"Terminated process with PID: {pid}")**

**self.show\_processes()**

**else:**

**#service\_info = psutil.win\_service\_get(identifier)**

**#print(identifier)**

**#os.system("sc config " + identifier + " start=disabled")**

**os.system(f'net stop {identifier}')**

**#os.system(f'net stop {}')**

**except Exception as e:**

**print(f"An error occurred while terminating the process/service: {e}")**

**def show\_services(self):**

**global a**

**a = 2**

**self.table\_widget.setColumnCount(4)**

**self.table\_widget.setHorizontalHeaderLabels(["Service Name", "Display Name", "Status", "Start Type"])**

**self.table\_widget.setRowCount(0)**

**try:**

**services = list(psutil.win\_service\_iter())**

**for service in services:**

**service\_name = service.name()**

**display\_name = service.display\_name()**

**status = service.status()**

**start\_type = service.start\_type()**

**row\_position = self.table\_widget.rowCount()**

**self.table\_widget.insertRow(row\_position)**

**self.table\_widget.setItem(row\_position, 0, QTableWidgetItem(service\_name))**

**self.table\_widget.setItem(row\_position, 1, QTableWidgetItem(display\_name))**

**self.table\_widget.setItem(row\_position, 2, QTableWidgetItem(status))**

**self.table\_widget.setItem(row\_position, 3, QTableWidgetItem(start\_type))**

**except Exception as e:**

**print(f"An error occurred while updating the services table: {e}")**

**def stop\_service(self):**

**# Add your code here to stop the selected service**

**pass**

**def show\_performance\_summary(self):**

**self.performance\_summary\_widget.update\_performance\_summary()**

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

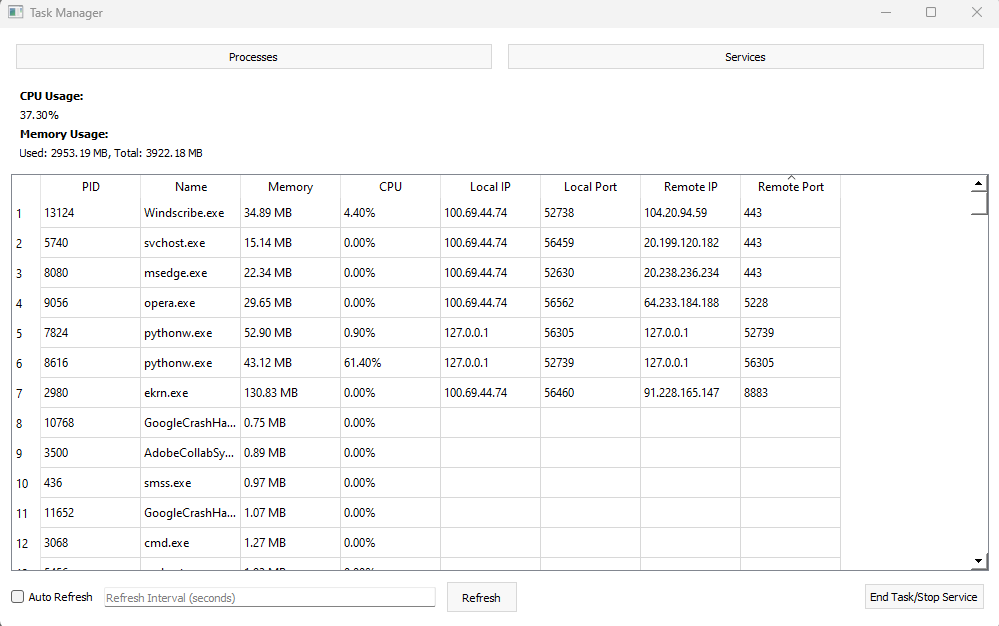
**app = QApplication(sys.argv)**

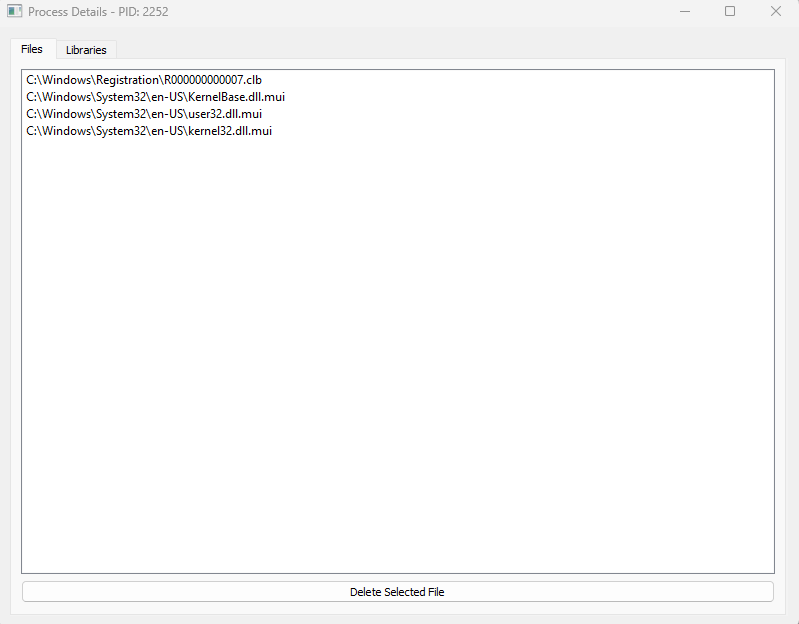
**window = TaskManagerApp()**

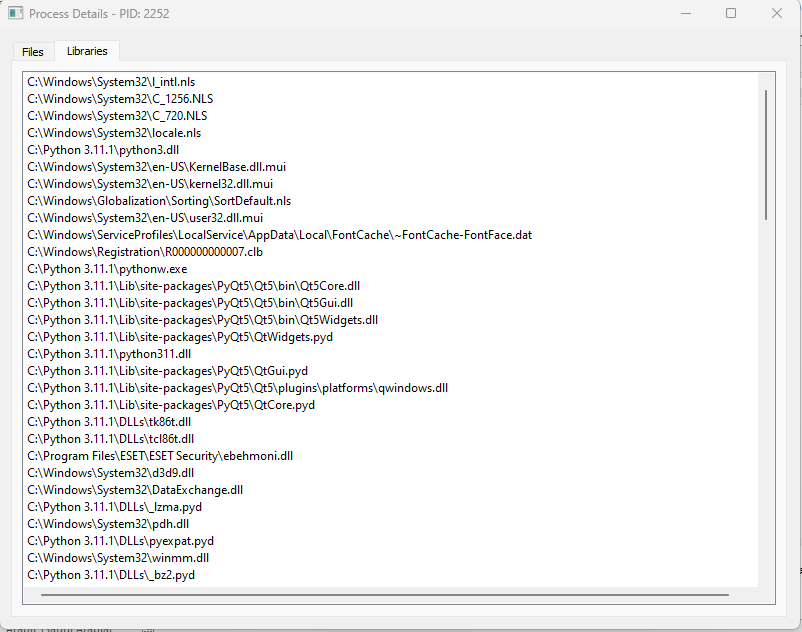
**window.show()**

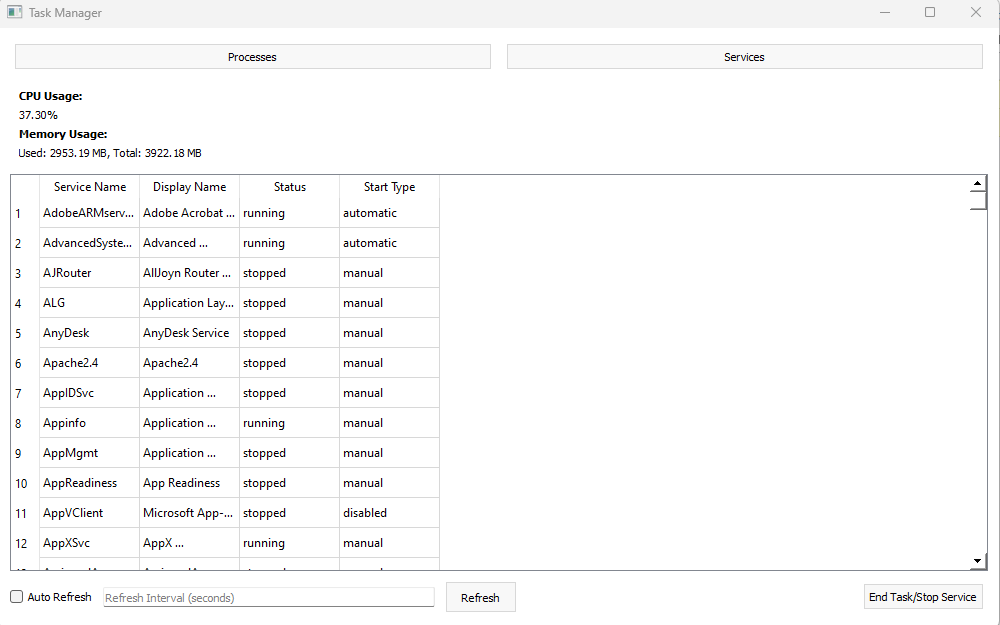
**sys.exit(app.exec\_())**

**صور من البرنامج:**

**1-واجهة العمليات**

**2-واجهة الملفات لكل عملية**

**3-واجهة المكاتب لكل عملية**

**4-واجهة الخدمات**

**الختام:**

برنامج إدارة المهام هذا يقدم واجهة مستخدم رسومية سهلة الاستخدام لمراقبة وإدارة العمليات الجارية على النظام. يستفيد البرنامج من مكتبتي Psutil و PyQt5 للوصول إلى معلومات النظام وإنشاء واجهة رسومية فعّالة.

يمكن توسيع البرنامج في المستقبل لتضمين المزيد من الميزات وتحسين أدائه وتجربة المستخدم.

Top of Form