



TRABAJO FINAL DE GRADO

JOB SCHEDULER

DESARROLLO DE UN SISTEMA DE
PROGRAMACIÓN DE TAREAS Y MONITOREO DE
RECURSOS EN LINUX

Autor: Marcos Zaragoza Ríos

ASIR

Curso 2023/24

IES LA VEREDA

Tutor: Clemente López Giner



Introducción

En la administración de sistemas operativos, la automatización de tareas y la monitorización de recursos son esenciales para garantizar la eficiencia y la continuidad del servicio. Este proyecto presenta el desarrollo de un Sistema de Programación y Monitoreo de Tareas para sistemas Linux, diseñado para simplificar y optimizar la gestión de tareas administrativas y de mantenimiento.

Inicialmente, la idea principal del proyecto era utilizar Apache Airflow para la gestión de tareas. Sin embargo, tras varios intentos fallidos de implementación debido a su complejidad y a las limitaciones encontradas en nuestro entorno específico, se decidió desarrollar desde cero una solución en Python.

El sistema, implementado en Python, permite definir y gestionar tareas automatizadas utilizando la biblioteca “schedule”. Además, se integra con la biblioteca “psutil” para monitorizar el uso de recursos del sistema, como la memoria y el espacio en disco. La configuración del sistema se realiza mediante un archivo JSON, lo que facilita su personalización y escalabilidad.

Las funcionalidades del sistema incluyen la creación de copias de seguridad, la limpieza periódica de la papelera de reciclaje, el envío de alertas por correo electrónico sobre el uso crítico de recursos y la generación de informes de usuarios. Los resultados obtenidos demuestran la efectividad del sistema en la automatización de tareas rutinarias y la supervisión continua de los recursos, contribuyendo así a la optimización de la administración del sistema.

Abstract

In system administration, task automation and resource monitoring are essential to ensure efficiency and service continuity. This project presents the development of a Task Scheduling and Monitoring System for Linux systems, designed to simplify and optimize the management of administrative and maintenance tasks.

Initially, the main idea of the project was to use Apache Airflow for task management. However, after several failed attempts to implement it due to its complexity and the limitations encountered in our specific environment, we decided to develop our own solution in Python.

The system, implemented in Python, allows defining and managing automated tasks using the "schedule" library. Additionally, it integrates with the "psutil" library to monitor system resource usage, such as memory and disk space. The system configuration is done through a JSON file, making it easy to customize and scalable.

The system's functionalities include creating backups, periodically cleaning the recycle bin, sending email alerts about critical resource usage, and generating user reports. The obtained results demonstrate the system's effectiveness in automating routine tasks and continuously monitoring resources, thus contributing to system administration optimization.

Resum

En l'administració de sistemes operatius, l'automatització de tasques i la monitorització de recursos són essencials per garantir l'eficiència i la continuïtat del servei. Aquest projecte presenta el desenvolupament d'un Sistema de Programació i Monitorització de Tasques per a sistemes Linux, dissenyat per simplificar i optimitzar la gestió de tasques administratives i de manteniment.

Inicialment, la idea principal del projecte era utilitzar Apache Airflow per a la gestió de tasques. No obstant això, després de diversos intents fallits d'implementació a causa de la seva complexitat i les limitacions trobades en el nostre entorn específic, es va decidir desenvolupar una solució pròpia en Python.

El sistema, implementat en Python, permet definir i gestionar tasques automatitzades utilitzant la biblioteca "schedule". A més, s'integra amb la biblioteca "psutil" per monitoritzar l'ús de recursos del sistema, com la memòria i l'espai en disc. La configuració del sistema es realitza mitjançant un fitxer JSON, cosa que facilita la seva personalització i escalabilitat.

Les funcionalitats del sistema inclouen la creació de còpies de seguretat, la neteja periòdica de la paperera de reciclatge, l'enviament d'alertes per correu electrònic sobre l'ús crític de recursos i la generació d'informes d'usuaris. Els resultats obtinguts demostren l'efectivitat del sistema en l'automatització de tasques rutinàries i la supervisió contínua dels recursos, contribuint així a l'optimització de l'administració del sistema.

Índice de contenidos

1. INTRODUCCIÓN.....	5
1.1 MÓDULOS A LOS QUE IMPLICA.....	5
1.2 BREVE DESCRIPCIÓN DEL PROYECTO.....	6
2. ESTUDIO PREVIO.....	7
Situación de la Empresa.....	7
Necesidades de la Elaboración del Proyecto.....	7
3. DISEÑO.....	8
3.1 DISEÑO GENERAL.....	8
3.2 DISEÑO DETALLADO.....	10
3.2.1 Job Scheduler.....	10
3.2.2 Configuración.....	10
3.2.3 Monitorización de Recursos.....	11
3.2.4 Envío de Correos Electrónicos.....	11
3.2.5 Limpieza de Papelera.....	11
4. IMPLANTACIÓN.....	12
4.1 Desarrollo de la aplicación.....	12
4.2 Procesos desarrollados.....	13
4.3 Impacto en el servidor y el cliente.....	13
5. RECURSOS.....	14
5.1 Herramientas hardware.....	14
5.2 Herramientas software.....	14
5.3 Sistemas operativos y paquetes necesarios.....	14
5.4 Personal.....	15
6. RESULTADOS Y PRUEBAS.....	15
6.1 Resultados Obtenidos.....	15
6.2 Análisis de Desempeño.....	15
6.3 Pruebas Realizadas.....	16
7. MANUAL DE USUARIO Y GUÍA DE INSTALACIÓN.....	17
7.1 Manual de Usuario.....	17
7.1.1 Estructura de Ficheros.....	17
7.2 Guía de instalación.....	30
8. CONCLUSIONES.....	32
8.1 Alcance y Logros.....	32
8.2 Desafíos, Problemas y Lecciones Aprendidas.....	32
8.3 Futuras Mejoras.....	33
8.4 Conclusión Final.....	33
9. REFERENCIAS.....	35
10. ANEXOS.....	36

1. INTRODUCCIÓN

1.1 MÓDULOS A LOS QUE IMPLICA

Este proyecto se basa en las habilidades y conocimientos adquiridos a lo largo de los cursos de primero y segundo de ASIR (Administración de Sistemas Informáticos en Red). A continuación, se detallan los módulos más relevantes y cómo sus contenidos han sido aplicados en el desarrollo de este trabajo:

- **Administración de Sistemas Operativos (ASO):** Este módulo ha sido fundamental para entender el entorno en el cual se ejecutan los trabajos programados, así como para manejar los comandos necesarios para la limpieza de archivos y monitoreo del sistema.
- **Servicios en Red (SER):** La configuración de servicios de red y la comprensión de protocolos de comunicación fueron esenciales para implementar la parte del proyecto que envía correos electrónicos con información relevante del sistema.
- **Aplicaciones Ofimáticas (AO):** Aunque de manera indirecta, las habilidades adquiridas en este módulo ayudaron en la documentación del proyecto y en la presentación de resultados de manera clara y profesional.
- **Administración de Sistemas Gestores de Bases de Datos (ASGBD):** Este conocimiento se utilizó para almacenar configuraciones y logs de las tareas programadas, asegurando que toda la información relevante esté disponible.
- **Seguridad y Alta Disponibilidad (SAD):** La comprensión de buenas prácticas de seguridad fue crucial para asegurar el entorno en el cual se desarrolló esta aplicación.

1.2 BREVE DESCRIPCIÓN DEL PROYECTO

Este proyecto es un Job Scheduler o más conocido como Programador de tareas, una herramienta diseñada para automatizar diversas tareas administrativas y de mantenimiento en un sistema. El objetivo principal es facilitar la ejecución de tareas repetitivas como la limpieza de la papelera, el envío de correos electrónicos con reportes del sistema, y la copia de seguridad de datos. Además, se busca proporcionar herramientas para monitorear el uso de memoria y disco, y alertar sobre posibles problemas antes de que afecten el rendimiento del sistema. La configuración flexible permite a los usuarios definir y configurar sus propias tareas programadas mediante un archivo de configuración sencillo.

El uso del proyecto es bastante intuitivo. Los usuarios pueden definir qué tareas desean automatizar, especificando la frecuencia y los detalles de cada tarea. El sistema se encarga de ejecutar las tareas programadas en los intervalos definidos, asegurando que las tareas críticas se realicen sin intervención manual. Además, ofrece funcionalidades para monitorear recursos del sistema y alertar a los administradores sobre posibles problemas, asegurando así una gestión proactiva del entorno informático.

En cuanto a las tecnologías investigadas, el proyecto está desarrollado principalmente en Python, aprovechando su versatilidad y la amplia disponibilidad de bibliotecas que facilitan la programación de tareas y la monitorización del sistema. Se utilizaron librerías como `schedule` para la programación de tareas y `psutil` para la monitorización de recursos del sistema. Además, se configuraron y utilizaron servidores de correo electrónico mediante SMTP para el envío automatizado de reportes y alertas, asegurando una comunicación eficiente y oportuna con los administradores del sistema.

2. ESTUDIO PREVIO

Situación de la Empresa

La empresa en la que se desarrolla este proyecto es de tamaño mediano y cuenta con una infraestructura de IT bastante compleja. Esta infraestructura incluye varios servidores, estaciones de trabajo y dispositivos de red que son esenciales para las operaciones diarias de la empresa. Con el crecimiento de la empresa, las demandas sobre su infraestructura de IT también han aumentado, creando una serie de necesidades específicas que este proyecto busca resolver.

Necesidades de la Elaboración del Proyecto

La empresa realiza diversas tareas de mantenimiento y administración que son repetitivas y consumen mucho tiempo. Estas tareas, como la limpieza de la papelera de reciclaje de los servidores, la realización de copias de seguridad y la generación de informes sobre el uso de recursos del sistema, actualmente se realizan de manera manual. Esto no solo consume tiempo, sino que también aumenta el riesgo de errores humanos. La automatización de estas tareas mediante un Job Scheduler reducirá significativamente la carga de trabajo del equipo de IT y mejorará la eficiencia de estas operaciones.

Además, la empresa necesita monitorizar el uso de recursos del sistema, como la memoria y el espacio en disco, y recibir alertas proactivas cuando estos recursos se están agotando. Esto es crucial para prevenir fallos del sistema que pueden afectar la productividad y la disponibilidad de servicios críticos.

Después de mi periodo de prácticas en el departamento de IT en “*Celestica*” me he dado cuenta que un sistema de alertas bien configurado permitirá al equipo reaccionar rápidamente ante posibles problemas, antes de que se conviertan en incidentes mayores.

3. DISEÑO

3.1 DISEÑO GENERAL

El diseño general del proyecto se centra en la estructura global del sistema y la interacción entre sus componentes principales. En nuestro caso, el sistema se compone de los siguientes módulos principales:

Job Scheduler: Este módulo es el núcleo del sistema y se encarga de la planificación y ejecución de las tareas programadas.

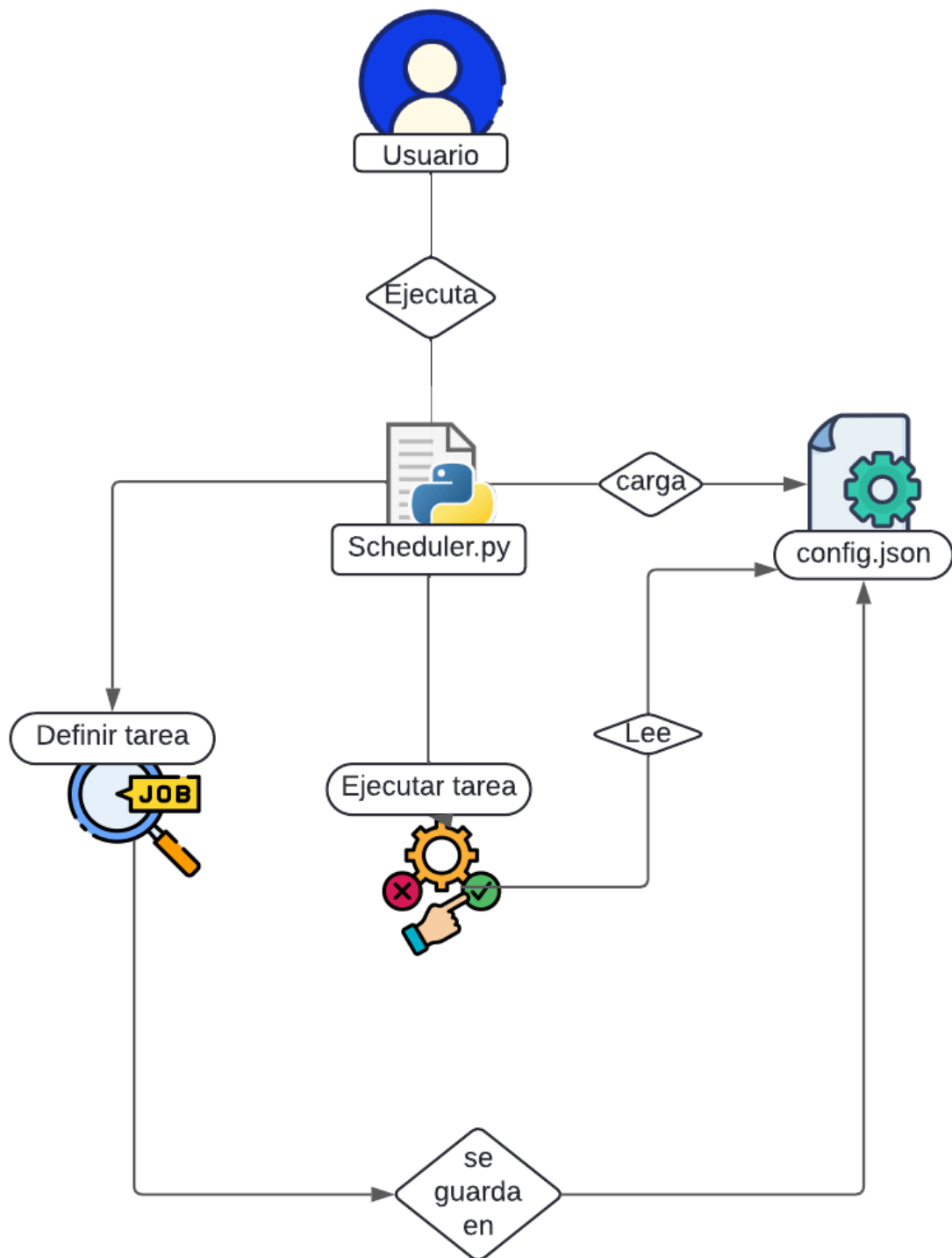
Configuración: Maneja la configuración del sistema, que incluye la definición de trabajos, intervalos de ejecución y otras opciones personalizadas.

Monitorización de Recursos: Supervisa el uso de recursos del sistema, como la memoria y el disco, para tomar decisiones informadas sobre la ejecución de tareas.

Envío de Correos Electrónicos: Gestiona el envío de correos electrónicos para notificar sobre el estado de las tareas programadas y otros eventos relevantes.

Limpieza de Papelera: Se encarga de limpiar la papelera del sistema, eliminando archivos que han sido marcados para su eliminación.

La interacción entre estos módulos se realiza a través de interfaces bien definidas, lo que permite una comunicación fluida y eficiente entre ellos.



3.2 DISEÑO DETALLADO

3.2.1 Job Scheduler

El módulo del Job Scheduler es el corazón del sistema, encargado de la planificación y ejecución de tareas programadas. Está compuesto por los siguientes elementos:

- **Planificador de tareas:** Utiliza un planificador de tareas para programar la ejecución de trabajos en base a intervalos de tiempo específicos, como diario, por hora o cada minuto.
- **Definición de trabajos:** Permite definir y gestionar los trabajos a ejecutar, especificando el tipo de tarea, el intervalo de ejecución y otros parámetros relevantes.
- **Ejecución de tareas:** Una vez programadas, las tareas son ejecutadas según el planificador de tareas y los parámetros definidos.

3.2.2 Configuración

La Configuración gestiona la configuración del sistema, permitiendo al usuario definir y personalizar diversos aspectos del funcionamiento del sistema.

Características principales:

- **Definición de Trabajos:** Permite al usuario definir nuevos trabajos, especificando el tipo de tarea, el intervalo de ejecución y otros parámetros
- **Configuración de Opciones:** Ofrece opciones de configuración avanzadas para personalizar el comportamiento del sistema, como la configuración del servidor SMTP que gestiona el envío de correos electrónicos.

3.2.3 Monitorización de Recursos

La parte de Monitorización de Recursos supervisa el uso de recursos del sistema, como la memoria y el disco, para garantizar un funcionamiento óptimo del sistema.

Características:

- **Obtención de Datos:** Utiliza bibliotecas y herramientas de monitorización para recopilar información sobre el uso de recursos del sistema.
- **Análisis de Datos:** Analiza los datos recopilados para identificar posibles cuellos de botella o situaciones de sobrecarga de hardware y recursos.

3.2.4 Envío de Correos Electrónicos

El módulo de envío de correos electrónicos se encarga de gestionar el envío de correos electrónicos para notificar sobre el estado de las tareas programadas y otros eventos relevantes.

Características:

- **Configuración de Correo:** Permite al usuario configurar los parámetros necesarios para el envío de correos electrónicos, como la dirección del servidor SMTP, el puerto y las credenciales de autenticación.
- **Generación de Mensajes:** Genera mensajes de correo electrónico personalizados para notificar sobre eventos específicos, como la finalización exitosa o fallida de una tarea.

3.2.5 Limpieza de Papelera

La limpieza de papelera se encarga de limpiar la papelera del sistema, eliminando archivos que han sido marcados para su eliminación.

Características principales:

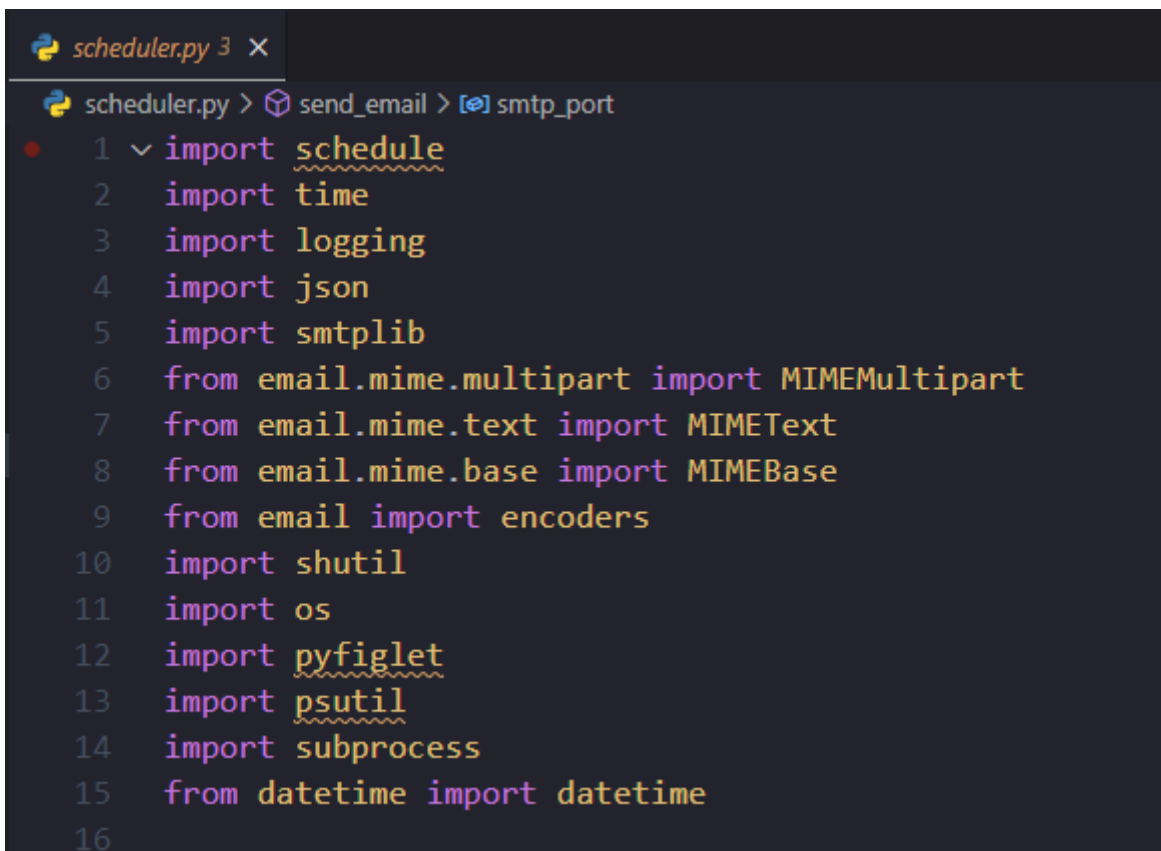
- **Exploración de la Papelera:** Escanea la papelera del sistema en busca de archivos que puedan ser eliminados de forma segura.

- Eliminación de Archivos: Elimina los archivos identificados de la papelera del sistema de manera permanente.

4. IMPLANTACIÓN

4.1 Desarrollo de la aplicación

Para el desarrollo del proyecto Job Scheduler, se utilizó un entorno de desarrollo basado en Python 3.12, ejecutado en un sistema operativo Ubuntu 24.04. Se empleó el IDE Visual Studio Code para la escritura y depuración del código, facilitando el desarrollo y la gestión de archivos del proyecto. Además, se hizo uso de diversas bibliotecas estándar de Python, como *json*, *logging*, *os*, *shutil*, *time*, *subprocess*, *sched*, y *smtplib*.



```
scheduler.py 3 X
scheduler.py > send_email > smtp_port
1  import schedule
2  import time
3  import logging
4  import json
5  import smtplib
6  from email.mime.multipart import MIMEMultipart
7  from email.mime.text import MIMEText
8  from email.mime.base import MIMEBase
9  from email import encoders
10 import shutil
11 import os
12 import pyfiglet
13 import psutil
14 import subprocess
15 from datetime import datetime
16
```

La configuración del entorno de desarrollo implicó la instalación de las bibliotecas mencionadas, junto con la creación de un entorno virtual de Python para gestionar las dependencias de manera aislada

4.2 Procesos desarrollados

El proceso de desarrollo del Job Scheduler se dividió en varias etapas:

1. Configuración inicial: Se realizó la configuración inicial del proyecto, incluyendo la carga de la configuración desde un archivo JSON, la definición de las funciones de tarea y la creación del bucle principal del programa.
2. Definición de trabajos: Se implementó la funcionalidad para que el usuario pueda definir nuevos trabajos, especificando el tipo de tarea, el intervalo de ejecución y los parámetros adicionales necesarios.
3. Programación de tareas: Se desarrolló la lógica para programar y ejecutar las tareas según el intervalo especificado por el usuario. Esto implicó el uso del módulo “*sched*” de Python para la gestión de eventos temporizados.
4. Monitorización de recursos: Se integró la monitorización del uso de memoria y disco en el sistema, permitiendo al usuario supervisar el rendimiento del sistema en tiempo real.

4.3 Impacto en el servidor y el cliente

El Job Scheduler tiene un impacto limitado tanto en el servidor como en el cliente. A nivel del servidor, el impacto se concentra en el consumo de recursos durante la ejecución de las tareas programadas, siendo este impacto mínimo debido a la optimización de los procesos implementados. En cuanto al cliente, la interfaz de usuario proporciona una experiencia intuitiva y fácil de usar, lo que minimiza la curva de aprendizaje y hace que el uso del Job Scheduler sea accesible para usuarios de todos los niveles.

5. RECURSOS

5.1 Herramientas hardware

El proyecto Job Scheduler se implementó en un entorno de desarrollo basado en el siguiente hardware. Se utilizó un portátil Lenovo con las siguientes especificaciones, aunque el proyecto en sí no necesita muchos recursos de hardware:

- Procesador: Intel Core i5 de 8ª generación
- Memoria RAM: 8 GB
- Almacenamiento: SSD de 256 GB
- Sistema operativo: Ubuntu 24.04 LTS

5.2 Herramientas software

Las herramientas de software utilizadas para el desarrollo e implementación del Job Scheduler incluyeron:

- Python 3.12: Lenguaje de programación principal utilizado para desarrollar la lógica del Job Scheduler.
- VS Code: Entorno de desarrollo integrado (IDE) utilizado para escribir, depurar y ejecutar el código Python.
- Bibliotecas estándar de Python: Se hicieron uso de diversas bibliotecas estándar de Python, como json, logging, os, shutil, time, subprocess, sched, y smtplib, entre otras.

5.3 Sistemas operativos y paquetes necesarios

El sistema operativo utilizado para el desarrollo y la implementación del Job Scheduler fue Ubuntu 24.04 LTS. Se seleccionó Ubuntu debido a su estabilidad, amplia disponibilidad de paquetes de software y su compatibilidad con Python y otras tecnologías utilizadas en el proyecto.

Los paquetes necesarios para la implementación del Job Scheduler se gestionaron mediante el sistema de gestión de paquetes de Python, pip. Se utilizó un entorno virtual de Python para gestionar las dependencias del proyecto de manera aislada.

5.4 Personal

El proyecto Job Scheduler fue desarrollado por un único desarrollador, Marcos Zaragoza, estudiante de segundo curso del ciclo formativo de grado superior en Administración de Sistemas Informáticos en Red (ASIR). Marcos Zaragoza también se encargó del mantenimiento y la gestión continua del proyecto durante su desarrollo.

6. RESULTADOS Y PRUEBAS

6.1 Resultados Obtenidos

El proyecto JobScheduler ha alcanzado los objetivos principales establecidos al inicio del desarrollo. El sistema es capaz de definir, programar y ejecutar diversas tareas de manera automática en un entorno Linux, lo cual incluye la capacidad de realizar copias de seguridad, enviar correos electrónicos con información relevante, monitorear el uso de recursos del sistema y limpiar la papelera del usuario.

6.2 Análisis de Desempeño

El desempeño del JobScheduler ha sido evaluado en términos de eficiencia y uso de recursos del sistema. Las tareas programadas se ejecutan de acuerdo con los intervalos especificados, y el impacto en el rendimiento del sistema es mínimo. La monitorización del uso de memoria y disco demuestra que el sistema puede gestionar múltiples tareas sin causar sobrecarga. Estos fueron algunos de los aspectos clave del análisis de desempeño:

- **Ejecución de Tareas:** Las tareas se ejecutan puntualmente sin retrasos significativos. La programación de tareas diarias, horarias y por minutos ha demostrado ser precisa y confiable.
- **Uso de CPU y Memoria:** Durante la ejecución de tareas intensivas como la creación de copias de seguridad, el uso de la CPU se incrementa temporalmente, pero vuelve a niveles normales una vez completada la tarea.
- **Impacto en el Sistema:** La carga adicional generada por el Job Scheduler es baja, asegurando que el sistema continúe operando sin interrupciones o ralentizaciones perceptibles.

6.3 Pruebas Realizadas

Se llevaron a cabo varias pruebas para asegurar la funcionalidad y fiabilidad del Job Scheduler. Estas pruebas incluyeron:

1. Pruebas de Funcionalidad:

- Definición y programación de tareas: Se verificó que el sistema permite definir y programar tareas correctamente, con diferentes intervalos y configuraciones.
- Ejecución de tareas: Se realizaron pruebas para asegurar que las tareas se ejecutan en los intervalos especificados y cumplen con su propósito (por ejemplo: creación de backups, envío de correos, limpieza de la papelera).

2. Pruebas de Integración:

- Integración con SMTP: Se probó el envío de correos electrónicos a través de un servidor SMTP configurado, asegurando la entrega de mensajes.
- Acceso a sistemas de archivos: Se verificó el acceso y manipulación de archivos del sistema, especialmente en la creación de copias de seguridad y limpieza de la papelera.

3. Pruebas de Desempeño:

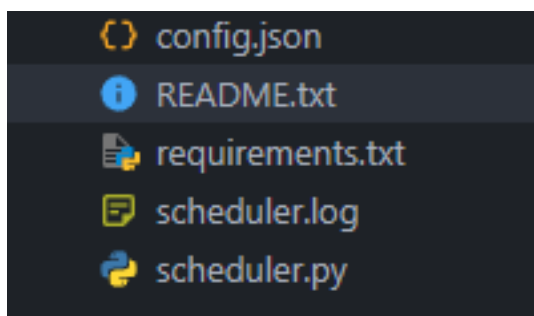
- Monitoreo de recursos: Se midió el uso de CPU y memoria durante la ejecución de tareas intensivas para evaluar el impacto en el sistema.
- Escalabilidad: Se probaron múltiples tareas simultáneamente para asegurar que el sistema puede manejar la carga sin degradación significativa en el rendimiento.

7. MANUAL DE USUARIO Y GUÍA DE INSTALACIÓN

7.1 Manual de Usuario

Este manual de usuario proporciona una guía detallada para utilizar el Job Scheduler. Se describen las funcionalidades principales, cómo configurarlo y cómo interpretar sus salidas.

7.1.1 Estructura de Ficheros



El directorio principal del Job Scheduler contiene los 5 ficheros principales:

- **config.json**

Este es el fichero de configuración principal, donde se especifican los detalles de los trabajos y la configuración del correo electrónico.

Este archivo debe incluir detalles como el servidor SMTP, credenciales de correo y trabajos a programar.

No tenemos que preocuparnos en configurar este fichero, ya que nuestro programa principal “jobscheduler.py” ya lo hace por nosotros.

A continuación se muestra un ejemplo de configuración:

```
config.json X
config.json > ...
1 {
2   "jobs": [
3     {
4       "task": "get_logged_in_users_and_send_email",
5       "interval": "minute",
6       "minutes": 30
7     },
8     {
9       "task": "clean_trash",
10      "interval": "hourly",
11      "minute": 30
12    },
13    {
14      "task": "backup_and_send_email",
15      "interval": "daily",
16      "time": "23:59"
17    }
18  ],
19  "email": {
20    "smtp_server": "smtp.gmail.com",
21    "smtp_port": 587,
22    "username": "marcoszaragoza2002@gmail.com",
23    "password": "wmas hyis mhem azai",
24    "from_addr": "marcoszaragoza2002@gmail.com",
25    "to_addrs": [
26      "tfgmarcosz@gmail.com"
27    ]
28  },
29  "daily_login_report": {
30    "hour": 23,
31    "minute": 59
32  },
33  "memory_alert_threshold": 25
34 }
```

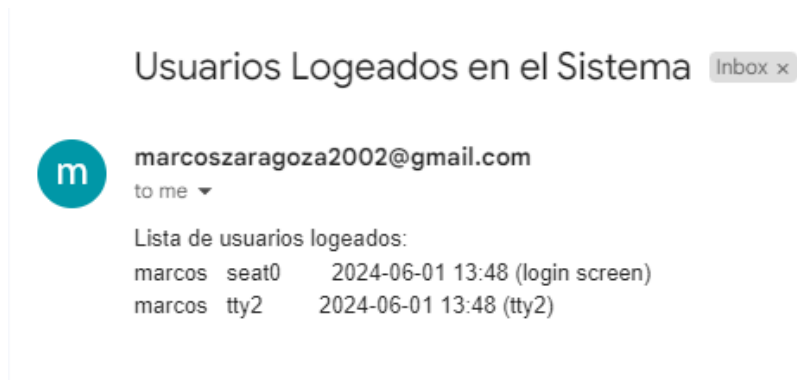
En este ejemplo se presentan tres jobs definidos,

- “get_logged_in_users_and_send_email”

Esta tarea analiza los usuarios que han iniciado sesión en nuestro sistema y se envía un mail al administrador con la siguiente información:

[Nombre de usuario que ha iniciado sesión]	[Terminal/Sesión]	[Fecha y hora de inicio de sesión]
--	-------------------	------------------------------------

Ejemplo del mail que nos llega a nuestra bandeja de entrada:



Esta tarea está programada para que se ejecute cada 30 minutos, tiene como finalidad mantener nuestro sistema controlado y monitorizado 24/7 para detectar accesos no autorizados a nuestro sistema.

- *“clean_trash”*

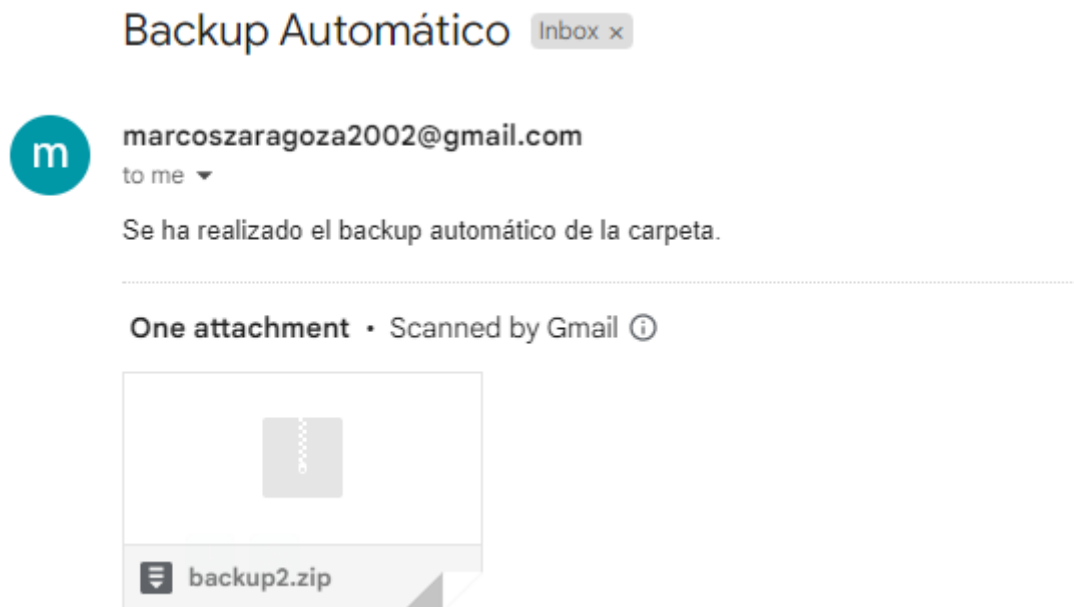
Esta tarea limpia la papelera de nuestro sistema, para eliminar archivos basura y liberar espacio, está programada cada hora y media.

Y el último,

- `"backup_and_send_email"`

Este job crea un backup diario de la carpeta que hayamos definido en nuestro programa principal `"scheduler.py"` nos la comprime y la envía por mail cada día a las 23:59 sin excepción, aunque vosotros podéis definir el intervalo en el que queráis que se ejecuten las tareas.

Así se vería un ejemplo del mail comprimido que nos llegará a nuestro correo:



[Email]

En este apartado se define la configuración del servidor SMTP (Simple Mail Transfer Protocol) que es el que se encargará de mandar los avisos mediante correos electrónicos.

Aquí hay que configurar los siguientes 6 parámetros:

Configuración del servidor SMTP	
<i>smtp_server</i>	Servidor smtp que usaremos, en nuestro caso el de google mail
<i>smtp_port</i>	Este parámetro define el puerto que se utilizará para conectarse al servidor SMTP. Los puertos comunes son 25, 465 y 587:
<i>username</i>	Este parámetro especifica el nombre de usuario que se utilizará para autenticar la conexión con el servidor SMTP. Generalmente, es la dirección de correo electrónico del remitente.
<i>password</i>	Este parámetro define la contraseña asociada con el nombre de usuario especificado. Se utiliza para la autenticación con el servidor SMTP.
<i>from_addr</i>	Este parámetro indica la dirección de correo electrónico del remitente. Es la dirección que aparecerá en el campo "De" (From) del correo enviado.
<i>to_addrs</i>	Este parámetro es una lista de direcciones de correo electrónico de los destinatarios. Especifica a quiénes se enviarán los correos electrónicos. Puede contener múltiples direcciones de correo.

Los parámetros *daily_login_report* y *memory_alert_threshold* son parámetros que se configurarán automáticamente.

El primero se usa para el envío de un informe diario sobre los usuarios que han iniciado sesión en el sistema.

Y el segundo define el umbral de uso de memoria (en porcentaje) que, al ser superado, desencadenará el envío de una alerta por correo electrónico.

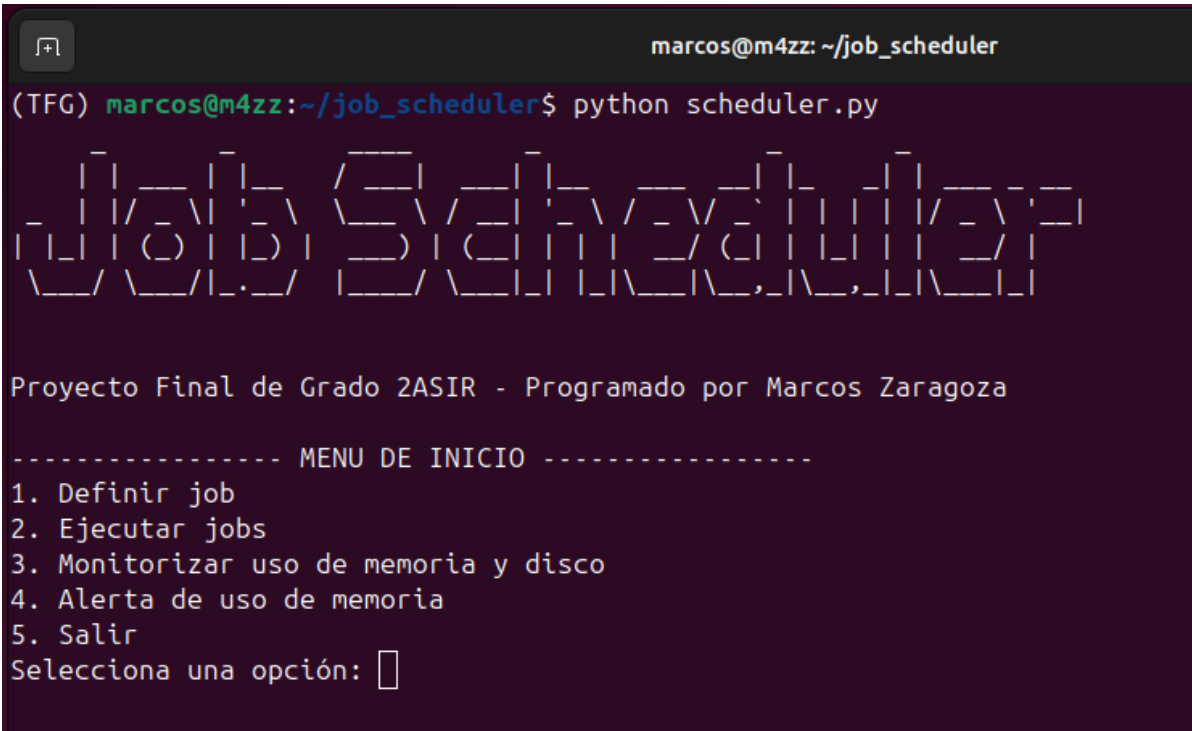
- scheduler.py

Es el script principal que contiene la lógica del Job Scheduler.

Este script está pensado para mantenerse ejecutado 24/7 en nuestro servidor.

Menú principal

Al ejecutar scheduler.py, se presenta un menú principal con las siguientes opciones:

A screenshot of a terminal window with a dark background. The terminal title bar shows 'marcos@m4zz: ~/job_scheduler'. The prompt is '(TFG) marcos@m4zz:~/job_scheduler\$' followed by the command 'python scheduler.py'. The output is a large ASCII art logo for 'Job Scheduler' in a light blue color. Below the logo, the text 'Proyecto Final de Grado 2ASIR - Programado por Marcos Zaragoza' is displayed. This is followed by a separator line '----- MENU DE INICIO -----'. A numbered list of five options is shown: '1. Definir job', '2. Ejecutar jobs', '3. Monitorizar uso de memoria y disco', '4. Alerta de uso de memoria', and '5. Salir'. At the bottom, the prompt 'Selecciona una opción:' is followed by a cursor icon (a small square).

1. Definir job: Permite agregar un nuevo trabajo a la programación.
2. Ejecutar jobs: Inicia la ejecución de los trabajos programados.
3. Monitorizar uso de memoria y disco: Monitorea continuamente el uso de memoria y disco.
4. Alerta de uso de memoria: Verifica el uso de memoria y envía una alerta si supera el umbral definido.
5. Salir: Salir del programa.

Definir un Job:

Seleccione la opción 1 en el menú principal para definir un nuevo trabajo. Se le presentarán las siguientes subopciones:

```
----- MENU DE INICIO -----
1. Definir job
2. Ejecutar jobs
3. Monitorizar uso de memoria y disco
4. Alerta de uso de memoria
5. Salir
Selecciona una opción: 1
Selecciona el tipo de job:
1. Backup y envío de correo electrónico
2. Monitorización de usuarios logeados
3. Programar Limpieza de Papelera
Elige una opción de job: 1
```

Backup y envío de correo electrónico: Define un trabajo para realizar un respaldo y enviar un correo.

Monitorización de usuarios logeados: Define un trabajo para enviar la lista de usuarios logeados.

Programar Limpieza de Papelera: Define un trabajo para limpiar la papelera.

Después de seleccionar una subopción, debe especificar el intervalo de ejecución (daily/hourly/minute) y la hora o minutos correspondientes.

```
Elige una opción de job: 1
Introduce el intervalo de ejecución (daily/hourly/minute): daily
Introduce la hora de ejecución diaria (HH:MM): 23:59
Job definido y guardado en config.json
```

Ejecutar Jobs

Seleccione la opción 2 en el menú principal para iniciar la ejecución de los trabajos programados. El sistema ejecutará las tareas en los intervalos definidos y registrará las actividades en el archivo de log.


```
----- MENU DE INICIO -----
1. Definir job
2. Ejecutar jobs
3. Monitorizar uso de memoria y disco
4. Alerta de uso de memoria
5. Salir
Selecciona una opción: 2
Jobs programados. Presiona Ctrl+C para detener.
Realizando backup y enviando correo...
Carpeta respaldada en /home/marcos/job_scheduler/backup2
Carpeta comprimida en /home/marcos/job_scheduler/backup2.zip
Correo enviado a ['tfgmarcosz@gmail.com']
█
```

Monitorizar Uso de Memoria y Disco

Seleccione la opción 3 para iniciar la monitorización continua del uso de memoria y disco. La información se mostrará en pantalla y se registrará en el archivo de log cada 60 segundos.

```
marcos@m4zz: ~/job_scheduler
(TFG) marcos@m4zz:~/job_scheduler$ python scheduler.py

Job Scheduler

Proyecto Final de Grado 2ASIR - Programado por Marcos Zaragoza

----- MENU DE INICIO -----
1. Definir job
2. Ejecutar jobs
3. Monitorizar uso de memoria y disco
4. Alerta de uso de memoria
5. Salir
Selecciona una opción: 3
Uso de memoria: 19.5%
Uso de disco: 6.3%
█
```

Alerta de Uso de Memoria

Seleccione la opción 4 para definir un umbral de alerta. Si el uso de memoria supera el umbral definido, se enviará una alerta por correo electrónico.

[illegible]

El umbral se guarda automáticamente en config.json:

```
16     },
17     "memory_alert_threshold": 15
18 }
```

En el ejemplo hemos definido un umbral de alerta cuando se supere el 15% de memoria. Como el uso de memoria es superior, se envía la alerta por correo, en caso de que no se supere, el programa se quedará ejecutado hasta que el umbral sea superior.


```
3185 2024-06-01 14:46:31,065 - INFO - Programando tareas
3186 2024-06-01 14:46:31,066 - INFO - Programando get_logged_in_users_and_send_email cada minute
3187 2024-06-01 14:47:48,210 - INFO - Correo de alerta enviado a ['tfgmarcosz@gmail.com']
3188 2024-06-01 14:48:07,767 - INFO - Correo de alerta enviado a ['tfgmarcosz@gmail.com']
3189 2024-06-01 14:48:46,552 - INFO - Uso de memoria: 30.7%
3190 2024-06-01 14:48:46,552 - INFO - Uso de disco: 6.3%
3191 2024-06-01 14:49:01,219 - INFO - Correo de alerta enviado a ['tfgmarcosz@gmail.com']
3192 2024-06-01 14:49:48,892 - INFO - Programando tareas
3193 2024-06-01 14:49:48,893 - INFO - Programando get_logged_in_users_and_send_email cada minute
3194 2024-06-01 14:49:48,893 - INFO - Programando clean_trash cada minute
3195 2024-06-01 14:50:50,483 - INFO - Correo enviado a ['tfgmarcosz@gmail.com']
3196 2024-06-01 14:50:50,484 - INFO - Papelera limpiada
3197 2024-06-01 14:51:51,720 - INFO - Correo enviado a ['tfgmarcosz@gmail.com']
3198 2024-06-01 14:51:51,721 - INFO - Papelera limpiada
3199 |
```

Este fichero ha sido crítico para la correcta programación de la aplicación, ya que gracias a él he detectado errores que hacían que mi programa no funcionara.

Este es un ejemplo de ERROR:

```
- INFO - Programando tareas
- INFO - Programando backup_and_send_email cada daily
- INFO - Programando get_logged_in_users_and_send_email cada daily
- ERROR - No se encontró una función válida para la tarea: get_logged_in_users_and_send_email
- INFO - Programando backup_and_send_email cada daily
- INFO - Programando backup_and_send_email cada daily
- INFO - Job: Backup Automático y Envío de Correo Electrónico
```

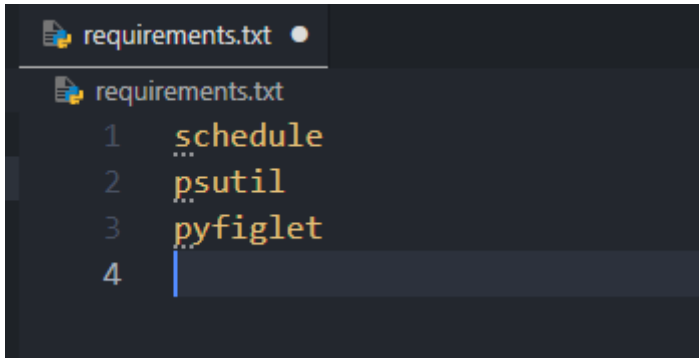
- README.txt

¿Qué es el archivo README.txt?

El archivo README.txt es un documento esencial que se incluye en la raíz de un proyecto de software. Su principal propósito es proporcionar información básica y crucial sobre el proyecto a cualquier persona que desee utilizarlo, desarrollarlo o contribuir a él. Es uno de los primeros archivos que los usuarios o desarrolladores consultan para comprender la naturaleza del proyecto, los requisitos para su uso, las instrucciones de instalación, y otros detalles importantes.

- requirements.txt

Este fichero contiene las librerías necesarias para el funcionamiento de nuestro Job Scheduler, en la guía de instalación explicaremos cómo instalarlas.

A screenshot of a code editor window with a dark theme. The title bar shows 'requirements.txt'. The editor content shows the same file name at the top, followed by a list of dependencies: 'schedule' on line 1, 'psutil' on line 2, 'pyfiglet' on line 3, and line 4 is empty with a blue cursor at the end. The text is color-coded: 'schedule' is orange, 'psutil' is yellow, and 'pyfiglet' is green.

```
requirements.txt
1  schedule
2  psutil
3  pyfiglet
4
```

En este fichero no se han incluido todas la librerías que usa nuestro programa, ya que las siguientes vienen instaladas por defecto con la instalación de Python:

```
os
json
shutil
smtpplib
subprocess
time
datetime
logging
zipfile
email.mime.multipart
email.mime.text
email.mime.base.
```

En caso de no estar instaladas, habría que instalarlas manualmente con el comando:

```
pip install [librería]
```

Solución de Problemas Comunes

Error al enviar correo: Verifique la configuración SMTP en config.json y asegúrese de que las credenciales y los servidores sean correctos.

Tarea no se ejecuta a la hora programada: Revise el archivo de log para identificar posibles errores y asegúrese de que el sistema esté en ejecución.

Uso de memoria alto sin alerta: Asegúrese de que el umbral de alerta esté correctamente configurado en config.json y que la opción 4 se haya ejecutado.

Este manual de usuario debe proporcionar toda la información necesaria para utilizar eficazmente el Job Scheduler. Si tiene preguntas adicionales o necesita asistencia técnica, consulte la documentación adicional o comuníquese con el correo de soporte tfgmarcosz@gmail.com.

7.2 Guía de instalación

Esta guía proporciona instrucciones detalladas para la instalación y configuración del Job Scheduler. Los archivos necesarios están alojados en un repositorio de GitHub, que debe ser clonado para completar el proceso de instalación.

Requisitos Previos

1. Sistema Operativo: Este programa está diseñado para funcionar en sistemas basados en Unix, como Linux.
2. Python: Asegúrese de tener Python 3.12 instalado en su sistema.
3. Git: Necesita Git instalado para clonar el repositorio.

Paso 1: Clonar el Repositorio de GitHub

Primero, debe clonar el repositorio que contiene el código del Job Scheduler. Abra una terminal y ejecute el siguiente comando:

```
git clone  
https://github.com/ieslavereda-projects/2324_ASIR_MARCOS_ZARAGOZA  
_RIOS_JOB_SCHEDULER.git
```

Esto creará una copia local del repositorio en su máquina.

Paso 2: Navegar al Directorio del Proyecto

Cambie al directorio del proyecto recién clonado:

```
cd job-scheduler
```

Paso 3: Crear y Activar un Entorno Virtual (opcional pero recomendado)

Para evitar conflictos de dependencias, se recomienda crear un entorno virtual para el proyecto:

```
python3 -m venv TFG
```

```
source TFG/bin/activate
```

Paso 4: Instalar Dependencias

Instale las dependencias necesarias utilizando pip:

```
pip install -r requirements.txt
```

Paso 5: Configurar el servidor SMTP

El servidor SMTP debe configurarse en el fichero de configuración config.json para definir los parámetros necesarios para el funcionamiento de envío de alerta del Job Scheduler.

En el Manual de usuario hay un apartado que explica todos los parámetros que pueden configurarse

Este es un ejemplo de configuración:

```
{
  "email": {
    "smtp_server": "smtp.example.com",
    "smtp_port": 587,
    "username": "usuario@example.com",
    "password": "contraseña_segura",
    "from_addr": "usuario@example.com",
    "to_addrs": ["destinatario1@example.com",
    "destinatario2@example.com"]
  },
}
```

Asegúrese de modificar estos valores según sus necesidades.

Paso 6: Ejecutar el Job Scheduler

Para ejecutar el Job Scheduler, utilice el siguiente comando:

```
python job_scheduler.py
```

Esto iniciará el programa y mostrará el menú de inicio, desde donde podrá definir jobs, ejecutar jobs, monitorizar el uso de recursos, y configurar alertas.

Siguiendo estos pasos, debería poder instalar y configurar el Job Scheduler correctamente. Si encuentra algún problema o tiene preguntas adicionales, no dude en consultar la documentación o contactar con el correo de soporte tfgmarcosz@gmail.com

8. CONCLUSIONES

El desarrollo del Job Scheduler ha sido un proyecto integral que ha permitido aplicar y consolidar mis conocimientos adquiridos a lo largo de la formación en los dos años de ASIR. Este proyecto ha cumplido con los objetivos planteados inicialmente y ha demostrado ser una herramienta eficaz para la automatización de tareas en entornos Linux.

8.1 Alcance y Logros

El principal logro del JobScheduler ha sido la implementación exitosa de un sistema que permite definir, programar y ejecutar tareas de manera automatizada.

8.2 Desafíos, Problemas y Lecciones Aprendidas

Durante el desarrollo del Job Scheduler, se enfrentaron varios desafíos que proporcionaron valiosas lecciones:

- Integración de servicios externos: La configuración y uso de servidores SMTP para el envío de correos electrónicos requirió una comprensión profunda de la seguridad y la configuración de red. Este proceso encontró problemas como la configuración incorrecta del servidor y las credenciales, que fueron resueltos mediante documentación exhaustiva, tutoriales en youtube y pruebas repetidas.
- Gestión de recursos: Implementar tareas que monitorean y utilizan recursos del sistema sin causar una sobrecarga fue un aspecto crítico. Se encontraron problemas de rendimiento cuando varias tareas se ejecutaban simultáneamente, lo que llevó a optimizar el código y mejorar la gestión de recursos.
- Manejo de excepciones: La importancia de manejar adecuadamente las excepciones y errores para asegurar la fiabilidad del sistema fue una lección clave. Durante las pruebas, se identificaron y resolvieron varios errores no manejados que podrían haber causado fallos en la ejecución de tareas.
- Compatibilidad y Permisos: Se encontraron problemas de compatibilidad con diferentes versiones de Linux y permisos de usuario que impedían la ejecución de ciertas tareas. Estos problemas se resolvieron ajustando las configuraciones y probando en diferentes entornos.

8.3 Futuras Mejoras

Aunque el Job Scheduler ha cumplido con los objetivos principales, siempre hay margen para mejoras y expansiones futuras:

- Interfaz de usuario: Desarrollar una interfaz gráfica de usuario para facilitar la definición y programación de tareas.
- Soporte para múltiples Sistemas Operativos: Extender la compatibilidad del Job Scheduler a otros sistemas operativos, como Windows y macOS.
- Notificaciones en tiempo real: Implementar un sistema de notificaciones en tiempo real para alertar a los usuarios sobre el estado de las tareas programadas y posibles fallos.

8.4 Conclusión Final

El Job Scheduler ha demostrado ser una solución efectiva y robusta para la automatización de tareas en entornos Linux. La implementación de este proyecto no solo ha proporcionado una herramienta útil para la gestión del sistema, sino que también ha permitido consolidar una serie de habilidades técnicas y de gestión de proyectos. El éxito de este proyecto sienta una base sólida para futuras exploraciones y desarrollos en el ámbito de la automatización y la gestión de sistemas.

9. REFERENCIAS

The Python Standard Library Documentation <https://docs.python.org/3/library/>

Schedule Library Documentation <https://schedule.readthedocs.io/en/stable/>

psutil Documentation <https://psutil.readthedocs.io/en/latest/>

schedule Documentation <https://schedule.readthedocs.io/en/stable/>

smtplib SMTP protocol client <https://docs.python.org/3/library/smtplib.html>

Python Email Library Documentation <https://docs.python.org/3/library/email.html>

Shutil High-level file operations <https://docs.python.org/3/library/shutil.html>

Real Python <https://realpython.com/>

Stack Overflow <https://stackoverflow.com/>

GitHub schedule library <https://github.com/dbader/schedule>

Scheduling Tasks Professionally in Python

<https://www.youtube.com/watch?v=yDPQfj4bZY8>

How To SCHEDULE Functions & Tasks In Python (FULL GUIDE)

<https://www.youtube.com/watch?v=FCPBG6NqMmQ>

Automatizar tareas con python

<https://www.iebschool.com/blog/automatizar-tareas-con-python-simplificando-tu-trabajo-diario-big-data/>

10. ANEXOS

Código Fuente Completo

Código fuente completo del proyecto, organizado y documentado para facilitar su comprensión y uso por otros desarrolladores.

scheduler.py (355 lineas)

```
1  #importar librerías
2  import schedule
3  import time
4  import logging
5  import json
6  import smtplib
7  from email.mime.multipart import MIMEMultipart
8  from email.mime.text import MIMEText
9  from email.mime.base import MIMEBase
10 from email import encoders
11 import shutil
12 import os
13 import pyfiglet
14 import psutil
15 import subprocess
16 from datetime import datetime
17 import zipfile
18
19 # configuración de logs
20 logging.basicConfig(filename='scheduler.log', level=logging.INFO,
21                     format='%(asctime)s - %(levelname)s - %(message)s')
22
23
24 # funcion para enviar el correo electrónico
25 def send_email(subject, body, attachment_path, config):
26     try:
27         # configuración del servidor SMTP, la cogemos del config.json
28         smtp_server = config['email']['smtp_server']
29         smtp_port = config['email']['smtp_port']
30         username = config['email']['username']
31         password = config['email']['password']
32         from_addr = config['email']['from_addr']
33         to_addrs = config['email']['to_addrs']
34
35         # crear el mensaje de correo
36         msg = MIMEMultipart()
37         msg['From'] = from_addr
38         msg['To'] = ", ".join(to_addrs)
39         msg['Subject'] = subject
40         msg.attach(MIMEText(body, 'plain'))
41
```

```
42     # adjuntar el archivo de backup si existe
43     if attachment_path and os.path.exists(attachment_path):
44         with open(attachment_path, "rb") as attachment:
45             part = MIMEBase('application', 'octet-stream')
46             part.set_payload(attachment.read())
47             encoders.encode_base64(part)
48             part.add_header('Content-Disposition', f"attachment;
49                             filename= {os.path.basename(attachment_path)}")
50             msg.attach(part)
51
52     # conectar al servidor SMTP y enviar el correo
53     server = smtplib.SMTP(smtp_server, smtp_port)
54     server.starttls()
55     server.login(username, password)
56     server.sendmail(from_addr, to_addrs, msg.as_string())
57     server.quit()
58
59     logging.info(f"Correo enviado a {to_addrs}")
60     print(f"Correo enviado a {to_addrs}")
61
62     except Exception as e:
63         logging.error(f"Error al enviar correo: {e}")
64         print(f"Error al enviar correo: {e}")
65
66
67 # funcion para realizar el backup de la carpeta
68 def backup_folder(source_folder, destination_folder):
69     try:
70         if os.path.exists(destination_folder):
71             shutil.rmtree(destination_folder)
72         shutil.copytree(source_folder, destination_folder)
73         logging.info(f"Carpeta respaldada en {destination_folder}")
74         print(f"Carpeta respaldada en {destination_folder}")
75         return True
76     except Exception as e:
77         logging.error(f"Error al realizar el backup: {e}")
78         print(f"Error al realizar el backup: {e}")
79         return False
80
81
82 # funcion para comprimir la carpeta de backup
83 def compress_backup_folder(folder_path, zip_path):
84     try:
85         with zipfile.ZipFile(zip_path, 'w') as zipf:
86             for root, dirs, files in os.walk(folder_path):
87                 for file in files:
88                     zipf.write(os.path.join(root, file),
89                               os.path.relpath(os.path.join(root, file), folder_path))
90         logging.info(f"Carpeta comprimida en {zip_path}")
91         print(f"Carpeta comprimida en {zip_path}")
92         return True
93     except Exception as e:
94         logging.error(f"Error al comprimir el backup: {e}")
95         print(f"Error al comprimir el backup: {e}")
96         return False
97
```

```
98
99 # funcion para cargar la configuración desde config.json
100 def load_config():
101     with open('config.json') as config_file:
102         return json.load(config_file)
103
104
105 # DEFINICION DE TAREAS
106 def backup_and_send_email():
107     logging.info("Job: Backup Automático y Envío de Correo Electrónico")
108     print("Realizando backup y enviando correo...")
109
110     source_folder = "/home/marcos/job_scheduler/prueba" # ruta de la carpeta a respaldar
111     destination_folder = "/home/marcos/job_scheduler/backup2" # ruta de la carpeta de destino para el backup
112     zip_path = "/home/marcos/job_scheduler/backup2.zip" # ruta del archivo comprimido de backup
113
114     # realizar el backup de la carpeta
115     if backup_folder(source_folder, destination_folder):
116         # comprimir la carpeta de backup
117         if compress_backup_folder(destination_folder, zip_path):
118             # enviar correo electronico con el archivo adjunto
119             subject = "Backup Automático"
120             body = "Se ha realizado el backup automático de la carpeta."
121             config = load_config() # cargar la configuración del correo
122             send_email(subject, body, zip_path, config)
123
124
125 def get_logged_in_users():
126     try:
127         # obtener la lista de usuarios logeados
128         logged_in_users = subprocess.check_output('who').decode('utf-8')
129         return logged_in_users
130     except Exception as e:
131         logging.error(f"Error al obtener la lista de usuarios logeados: {e}")
132         print(f"Error al obtener la lista de usuarios logeados: {e}")
133         return ""
134
135
136 def get_logged_in_users_and_send_email():
137     try:
138         config = load_config()
139         users = get_logged_in_users()
140         subject = "Usuarios Logeados en el Sistema"
141         body = f"Lista de usuarios logeados:\n{users}"
142         send_email(subject, body, None, config)
143     except Exception as e:
144         logging.error(f"Error al enviar la lista de usuarios logeados: {e}")
145         print(f"Error al enviar la lista de usuarios logeados: {e}")
146
```

```

148 def clean_trash():
149     try:
150         trash_dir = os.path.expanduser('~/.local/share/Trash/files')
151         if os.path.exists(trash_dir):
152             for filename in os.listdir(trash_dir):
153                 file_path = os.path.join(trash_dir, filename)
154                 try:
155                     if os.path.isfile(file_path) or os.path.islink(file_path):
156                         os.unlink(file_path)
157                     elif os.path.isdir(file_path):
158                         shutil.rmtree(file_path)
159                 except Exception as e:
160                     logging.error(f"Error al eliminar {file_path}: {e}")
161                     print(f"Error al eliminar {file_path}: {e}")
162             logging.info("Papelera limpiada")
163             print("Papelera limpiada")
164         else:
165             logging.info("La carpeta de la papelera no existe")
166             print("La carpeta de la papelera no existe")
167     except Exception as e:
168         logging.error(f"Error al limpiar la papelera: {e}")
169         print(f"Error al limpiar la papelera: {e}")
170
171
172 # definir un diccionario para mapear nombres de tareas a funciones
173 task_functions = {
174     'backup_and_send_email': backup_and_send_email,
175     'get_logged_in_users_and_send_email': get_logged_in_users_and_send_email,
176     'clean_trash': clean_trash
177 }
178
179
180 def schedule_jobs(config):
181     logging.info("Programando tareas")
182     for job in config['jobs']:
183         logging.info(f"Programando {job['task']} cada {job['interval']}")
184         task_function = task_functions.get(job['task'])
185         if task_function and callable(task_function):
186             if job['interval'] == 'daily':
187                 schedule.every().day.at(job['time']).do(task_function)
188             elif job['interval'] == 'hourly':
189                 schedule.every().hour.at(f"{job['minute']}").do(task_function)
190             elif job['interval'] == 'minute':
191                 schedule.every(job['minutes']).minutes.do(task_function)
192         else:
193             logging.error(f"No se encontró una función válida para la tarea: {job['task']}")
194
195 # menu
196 def display_menu():
197     print("----- MENU DE INICIO -----")
198     print("1. Definir job")
199     print("2. Ejecutar jobs")
200     print("3. Monitorizar uso de memoria y disco")
201     print("4. Alerta de uso de memoria")
202     print("5. Salir")
203

```



```

205 def display_task_menu():
206     print("Selecciona el tipo de job:")
207     print("1. Backup y envío de correo electrónico")
208     print("2. Monitorización de usuarios logeados")
209     print("3. Programar Limpieza de Papelera")
210
211
212 def define_job(config):
213     display_task_menu()
214     task_choice = input("Elige una opción de job: ")
215
216     if task_choice == '1':
217         task = 'backup_and_send_email'
218     elif task_choice == '2':
219         task = 'get_logged_in_users_and_send_email' # define el nuevo job para obtener
220     elif task_choice == '3': # la lista de usuarios y enviar el correo
221         task = 'clean_trash' # limpieza de papelera
222     else:
223         print("Opción de job no válida")
224         return
225
226     interval = input("Introduce el intervalo de ejecución (daily/hourly/minute): ")
227
228     if interval == 'minute':
229         minutes = int(input("Introduce el número de minutos: "))
230         job = {"task": task, "interval": interval, "minutes": minutes}
231     elif interval == 'hourly':
232         minute = int(input("Introduce el minuto de la hora (0-59): "))
233         job = {"task": task, "interval": interval, "minute": minute}
234     elif interval == 'daily':
235         time = input("Introduce la hora de ejecución diaria (HH:MM): ")
236         job = {"task": task, "interval": interval, "time": time}
237     else:
238         print("Intervalo no válido")
239         return
240
241     config['jobs'].append(job)
242     with open('config.json', 'w') as config_file:
243         json.dump(config, config_file, indent=4)
244     print("Job definido y guardado en config.json")
245
246
247 def monitor_memory_and_disk_usage():
248     try:
249         while True:
250             memory = psutil.virtual_memory()
251             disk = psutil.disk_usage('/')
252             logging.info(f"Uso de memoria: {memory.percent}%")
253             logging.info(f"Uso de disco: {disk.percent}%")
254             print(f"Uso de memoria: {memory.percent}%")
255             print(f"Uso de disco: {disk.percent}%")
256             time.sleep(60)
257     except KeyboardInterrupt:
258         print("Monitorización interrumpida")
259

```

```

260
261 def send_alert_email(subject, body, config):
262     try:
263         # configuracion servidor SMTP para envío de alertra
264         smtp_server = config['email']['smtp_server']
265         smtp_port = config['email']['smtp_port']
266         username = config['email']['username']
267         password = config['email']['password']
268         from_addr = config['email']['from_addr']
269         to_addrs = config['email']['to_addrs']
270
271         # crear el mensaje de correo
272         msg = MIMEMultipart()
273         msg['From'] = from_addr
274         msg['To'] = ", ".join(to_addrs)
275         msg['Subject'] = subject
276         msg.attach(MIMEText(body, 'plain'))
277
278         # conectar al servidor SMTP y enviar el correo
279         server = smtplib.SMTP(smtp_server, smtp_port)
280         server.starttls()
281         server.login(username, password)
282         server.sendmail(from_addr, to_addrs, msg.as_string())
283         server.quit()
284
285         logging.info(f"Correo de alerta enviado a {to_addrs}")
286         print(f"Correo de alerta enviado a {to_addrs}")
287
288     except Exception as e:
289         logging.error(f"Error al enviar el correo de alerta: {e}")
290         print(f"Error al enviar el correo de alerta: {e}")
291
292
293 def check_memory_usage_and_alert(config):
294     try:
295         # verifica si el umbral de alerta de memoria esta en la configuracion
296         if 'memory_alert_threshold' not in config:
297             # si no esta pregunta al usuario por el umbral
298             memory_threshold = int(input("Introduce el umbral de alerta de memoria (en porcentaje): "))
299             config['memory_alert_threshold'] = memory_threshold
300
301             # guardar la configuracion actualizada en config.json
302             with open('config.json', 'w') as config_file:
303                 json.dump(config, config_file, indent=4)
304         else:
305             # si esta obtener el umbral de la configuracion
306             memory_threshold = config['memory_alert_threshold']
307
308             # obtener el uso de memoria
309             memory_usage = psutil.virtual_memory()
310             memory_percent = memory_usage.percent
311
312             # verificar si se supera el umbral de alerta
313             if memory_percent > memory_threshold:
314                 subject = "Alerta de Uso de Memoria"
315                 body = f"El uso de memoria ha superado el umbral de {memory_threshold}%. Uso actual: {memory_percent}%."
316                 send_alert_email(subject, body, config)
317

```

```
318 ~ except Exception as e:
319 ~     logging.error(f"Error al verificar el uso de memoria: {e}")
320 ~     print(f"Error al verificar el uso de memoria: {e}")
321
322
323
324 ~ def main():
325 ~     ascii_banner = pyfiglet.figlet_format("Job Scheduler")
326 ~     print(ascii_banner)
327
328 ~     # carga la configuracion desde config.json
329 ~     config = load_config()
330
331 ~     while True:
332 ~         display_menu()
333 ~         choice = input("Selecciona una opción: ")
334
335 ~         if choice == '1':
336 ~             define_job(config)
337 ~         elif choice == '2':
338 ~             schedule_jobs(config)
339 ~             print("Jobs programados. Presiona Ctrl+C para detener.")
340 ~             while True:
341 ~                 schedule.run_pending()
342 ~                 time.sleep(1)
343 ~         elif choice == '3':
344 ~             monitor_memory_and_disk_usage()
345 ~         elif choice == '4':
346 ~             check_memory_usage_and_alert(config)
347 ~         elif choice == '5':
348 ~             print("Saliendo...")
349 ~             break
350 ~         else:
351 ~             print("Opción no válida. Inténtalo de nuevo.")
352
353
354 ~ if __name__ == "__main__":
355 ~     main()
356
```

Repositorio GitHub

Junto a esta memoria adjunto un enlace a un repositorio de GitHub, donde están todos los ficheros y configuraciones necesarios para el funcionamiento del Job Scheduler.

https://github.com/ieslavereda-projects/2324_ASIR_MARCOS_ZARAGOZA_RIOS_JOB_SCHEDULER