

1IEE14 - Laboratorio 4

Instrucciones para el laboratorio:

- Materiales permitidos: Wiki del curso, apuntes de clase, consultar foros, tutoriales o documentación de Python online.
- Está prohibido el uso de cualquier modelo de lenguaje como ChatGPT o usar Github Copilot. A cualquier alumno que se le detecte que ha consultado un modelo de lenguaje se le pondrá nota 0(cero) en el laboratorio.
- Usted debe subir a Paideia un solo archivo comprimido (.zip o .rar) con el nombre L4_CODIGOPUCP.zip o L4_CODIGOPUCP.rar el cual contendrá sus archivos de Python para cada pregunta. En caso se le pida responder de manera teórica o incluir capturas de pantalla, deberá adjuntar un documento PDF con sus respuestas.
- El horario máximo permitido para subir el archivo es a las 10:00:00 pm. Pasada esa hora, habrá una penalidad de 2 puntos por cada minuto extra que se demore en entregar su archivo.

Pregunta 1: Monitoreo de temperatura (8 puntos)

Diseñar e implementar un sistema de tres servidores en Python que simulen el monitoreo de la temperatura de una máquina industrial (sensores entre 10 °C y 40 °C), usando comunicaciones TCP, flags de bloqueo, procesamiento bloqueante y generación de alertas basadas en métricas.

1. Servidor B (server_B.py)

- **Puerto:** escucha en *localhost:8001*
- **Recepción de datos:** cada 0.5s llega una línea de texto con un timestamp y la temperatura (YYYY-MM-DDTHH:MM:SS.xxx|<temp>\n) desde A. Hint: Se sugiere utilizar *socket.socket()* y *conn.makefile('r')*.

Ejemplo: 2025-04-24T13:05:31.260348| 23.57

- **Acumulación:** almacena cada lectura en una lista; al llegar a 10 valores:
 - i. Se conecta a **A** en el puerto de flags (*9000*) y envía **BUSY:B**.
 - ii. Simula un procesamiento bloqueante que aleatoriamente espera entre 1 y 3 s.
 - iii. Calcula la **media** de las 10 temperaturas.
 - iv. Crea el archivo `results/B_metrics_<YYYYMMDD_HHMMSS>.txt` cuyo contenido es la media con 4 decimales.
 - v. Notifica a **A** su liberación ejecutando **FREE:B**.
 - vi. Vuelve a comenzar el ciclo de acumulación.

2. Servidor C (server_C.py)

- **Puerto:** escucha en *localhost:8002*
- Funciona idéntico a B, salvo que calcula la varianza muestral de las 10 lecturas y la escribe en `results/C_metrics_<YYYYMMDD_HHMMSS>.txt`.

3. Servidor A (server_A.py):

- **Ejecución:**

```
user@vm:~/lab4/preg1$ python3 organizer.py M N
```

donde **M** y **N** son los umbrales de alerta y deben ser ingresados por línea de comandos.

- **Envío de lecturas:** cada 0.5s genera una temperatura aleatoria *float* entre 10.0 y 40.0 °C, con timestamp, y la envía **simultáneamente** a B y C.
- **Control de flujo:** abre un socket en *0.0.0.0:9000* para recibir flags **BUSY:<srv>** y **FREE:<srv>**. Mientras **B** o **C** estén en estado **BUSY**, **A** debe pausar el envío. Solo continua cuando **ambos** estén **FREE**.
- **Monitoreo de archivos:** cada 1s inspecciona el directorio *results/* y **solo** procesa nuevos archivos que cumplan los patrones:
 - *B_metrics_*.txt*
 - *C_metrics_*.txt*
- **Generación de alertas e informe crítico:**
 - Al detectar *B_metrics_<fecha>.txt*, lee la media. Si $media > M$, imprime:
`[A]<timestamp> ALERTA: Media elevada: <media> > <M>`
 - Al detectar *C_metrics_<fecha>.txt*, lee la varianza; si $varianza > N$, imprime:
`[A]<timestamp> ALERTA: Varianza elevada: <varianza> > <N>`
 - Si para la misma fecha ambas condiciones se cumplen, y aún no existe alerta crítica para esa fecha, crea en *results/* el archivo:
`WARNING_CRITICAL_FAILURE_<fecha>.txt`
 con contenido:
`CRITICAL FAILURE at <fecha>`
`media=<media>; varianza=<varianza>`
 e imprime en la consola:
`[A]<timestamp> *** WARNING CRITICAL GENERATED:`
`WARNING_CRITICAL_FAILURE_<fecha>.txt`

Importante

1. Si antes habías arrancado alguno de los servidores y los puertos quedan ocupados, libera los puertos con:
2. Si desea ejecutar los 3 servidores en una misma ventana de comandos, lanza los procesos **en este orden:**

```
user@vm:~/lab4/preg1$ sudo fuser -k 8001/tcp 8002/tcp 9000/tcp
user@vm:~/lab4/preg1$ # 1) Servidor B en background
python3 server_B.py &
# 2) Servidor C en background
python3 server_C.py &
# 3) Espera un segundo para asegurar binds() correctos
sleep 1
# 4) Servidor A en primer plano, con umbrales M=20, N=5
python3 server_A.py 20 5
```

Sugerencias

- **Sockets:** `socket.socket()`, `bind()`, `listen()`, `accept()`, `connect()`, `sendall()`, `makefile('r')`
- **Concurrencia:** `threading.Thread()`, `threading.Lock()`
- **Tiempo:** `time.sleep()`, `random.uniform()`
- **Fechas:** `datetime.datetime.now()`, `strftime()`, `fromisoformat()`
- **Filesystem:** `os.makedirs()`, `os.listdir()`, `open(...)`

- **Cálculos:** `statistics.mean()`, `statistics.pvariance()`
- **Argumentos:** `sys.argv`

Entregables

- Los tres scripts (`server_A.py`, `server_B.py`, `server_C.py`) debidamente comentados.
- Capturas de pantalla que muestren las alertas y la alerta crítica generada utilizando `M=20` y `N=5` (PDF dentro de su archivo ZIP o RAR).
- El contenido de al menos un `WARNING_CRITICAL_FAILURE_*.txt`.

Salida esperada

```

joant@LAPTOP-KAHN9ACA:~/lab4/preg1$ # 1) Servidor B en background
python3 server_B.py &
# 2) Servidor C en background
python3 server_C.py &
# 3) Espera un segundo para asegurar binds() correctos
sleep 1
# 4) Servidor A en primer plano, con umbrales M=20, N=5
python3 server_A.py 20 5
[1] 9759
[2] 9760
[B] Escuchando en 8001
[C] Escuchando en 8002
[B] Conexión establecida con A
[C] Conexión establecida con A
[B] Recibido temp=15.43
[C] Recibido temp=15.43
[A][2025-04-24T16:13:45.385730] Enviado temp=15.43
[A][2025-04-24T16:13:45.886014] Enviado temp=37.14
[C] Recibido temp=37.14
[B] Recibido temp=37.14
[A][2025-04-24T16:13:46.386256] Enviado temp=29.67
[C] Recibido temp=29.67
[B] Recibido temp=29.67
[A][2025-04-24T16:13:46.886530] Enviado temp=12.66
[C] Recibido temp=12.66
[B] Recibido temp=12.66
[A][2025-04-24T16:13:47.386725] Enviado temp=23.65
[B] Recibido temp=23.65
[C] Recibido temp=23.65
[A][2025-04-24T16:13:47.886888] Enviado temp=26.26
[B] Recibido temp=26.26
[C] Recibido temp=26.26
[A][2025-04-24T16:13:48.387233] Enviado temp=29.01
[C] Recibido temp=29.01
[B] Recibido temp=29.01
[A][2025-04-24T16:13:48.887550] Enviado temp=39.58
[B] Recibido temp=39.58
[C] Recibido temp=39.58
[A][2025-04-24T16:13:49.387728] Enviado temp=34.01
[B] Recibido temp=34.01
[C] Recibido temp=34.01
[A][2025-04-24T16:13:49.888172] Enviado temp=24.1
[C] Recibido temp=24.1
[B] Recibido temp=24.1
[A][2025-04-24T16:13:49.888537] Flag BUSY de B
[A][2025-04-24T16:13:49.888583] Flag BUSY de C
[A][2025-04-24T16:13:49.888887] ALERTA: Varianza elevada: 68.08 > 5.0
[A][2025-04-24T16:13:49.889122] ALERTA: Media elevada: 27.15 > 20.0
[A][2025-04-24T16:13:49.889243] *** WARNING CRITICAL GENERATED: WARNING_CRITICAL_FAILURE_20250424_161349.txt
[A][2025-04-24T16:13:52.890837] Flag FREE de B
[A][2025-04-24T16:14:02.604682] Flag FREE de C
[B] Recibido temp=10.84
[C] Recibido temp=10.84
  
```

Criterios de evaluación

- [1.0 pts] Comunicación TCP correcta
- [2.0 pts] Flags BUSY/FREE y lógica de pausa/envío en `server_A`
- [1.0 pts] Generación de alertas (console) y archivo crítico
- [2.0 pts] Monitoreo e I/O de archivos (`results/`, nombres y lectura segura)
- [2.0 pts] Obtención correcta de la salida esperada

Pregunta 2: Análisis de crecimiento financiero (5 puntos)

Dispones de los tres archivos CSV (`growth_A.csv`, `growth_B.csv`, `growth_C.csv`), cada uno con este formato:

```
month,revenue
2024-01,10500.50
2024-02,11320.00
...
```

[1.0 pts] Parte A: `analyze_growth.py`

- Ejecución:
`user@vm:~/lab4/preg1$ python3 analyze_growth.py growth_A.csv`
- Lee un único CSV (p.ej. `growth_A.csv`) usando el módulo `csv`.
- Convierte `revenue` a float y almacena meses y facturación en listas.
- Calcula:
 - Crecimiento porcentual mensual: $(\text{rev}[i] - \text{rev}[i-1]) / \text{rev}[i-1] \times 100$ for $i=1...11$.
 - Promedio de crecimiento: usar `statistics.mean()`
 - Desviación estándar de los crecimientos: usar `statistics.pstdev()`
 - Los 3 meses con mayor crecimiento: ordena la lista de crecimientos y toma los índices `top3`.
- Crea la función **`getStats`** que entregue un diccionario con los siguientes campos: `month`, `monthly_growth`, `avg_growth`, `std_growth`, `top3`
- Escribe `growth_stats.csv` utilizando **`getStats`** con columnas:
`month,monthly_growth,avg_growth,std_growth,top3`
Ejemplo de una fila:
`2024-02,7.81,5.23,2.14,['2024-05','2024-08','2024-12']`

Salida esperada

```
● joant@LAPTOP-KAHN9ACA:~/lab4/preg2$ python3 analyze_growth.py growth_A.csv
Escrito growth_stats.csv con 99 filas.
```

[2.0 pts] Parte B: `analyze_growth_sync.py` (Sincrono)

- Ejecución:
`user@vm:~/lab4/preg1$ python3 analyze_growth_sync.py growth_A.csv growth_B.csv growth_C.csv`
- Importa la función **`getStats`**.
- Para los tres archivos (`growth_A.csv`, `growth_B.csv`, `growth_C.csv`):
 1. Llama a esa función secuencialmente.
 2. Almacena cada resultado en una lista de dicts.
- Una vez terminados, escribe un único CSV `all_growth_stats_sync.csv` con las filas de las tres fuentes.
- Mide el tiempo total de ejecución con `time.perf_counter()` e imprime en consola:
`Tiempo total (sincrono): X.XXXX s`

Salida esperada

```
joant@LAPTOP-KAHN9ACA:~/lab4/preg2$ python3 analyze_growth_sync.py growth_A.csv growth_B.csv growth_C.csv
Tiempo total (sincrono): 0.0018 s
Escrito all_growth_stats_sync.csv con 297 filas.
```

[2.0 pts] Parte C: analyze_growth_async.py (Asíncrono)

- Ejecución:

```
user@vm:~/lab4/preg1$ python3 analyze_growth_async.py growth_A.csv growth_B.csv growth_C.csv
```

- Usa **asyncio** y **paraleliza** la lectura y el cálculo para los tres archivos. Opciones: *asyncio.to_thread()* para la función bloqueante de análisis, o *aiofiles* + *asyncio.gather()*.
- Lanza las tres tareas en paralelo con *asyncio.gather()*.
- Una vez finalizadas, consolida los resultados en *all_growth_stats_async.csv* con mismo formato que el sincronizado.
- Mide el tiempo total de ejecución con *time.perf_counter()* y muestra:

```
Tiempo total (Asíncrono): Y.YYYY s
```

Salida esperada

```
joant@LAPTOP-KAHN9ACA:~/lab4/preg2$ python3 analyze_growth_async.py growth_A.csv growth_B.csv growth_C.csv
Tiempo total (Asíncrono): 0.0024 s
Escrito all_growth_stats_async.csv con 297 filas.
```

Sugerencias de funciones Python

- **Lectura CSV:** *csv.reader()*, *open(..., newline="")*
- **Cálculos:** *statistics.mean()*, *statistics.pstdev()*
- **Ordenación y top-n:** *sorted(zip(...), key=lambda x: x[1], reverse=True)[:3]*
- **Fechas:** manipular cadenas, no necesario *datetime* aquí
- **Sincronía vs. Asíncrona:**
 - *time.perf_counter()*
 - *asyncio.run()*, *async def*, *await*, *asyncio.gather()*, *asyncio.to_thread()*
 - (Opcional) *aiofiles.open()* para I/O no bloqueante

Entregables

- Los scripts (*analyze_growth.py*, *analyze_growth_sync.py*, *analyze_growth_async.py*) debidamente comentados.
- Los archivos *csv* *growth_stats.csv*, *all_growth_stats_sync.csv* y *all_growth_stats_async.csv*.

Pregunta 3: Gestión de inventario y transacciones (7 puntos)

Dispones de un archivo **inventory.csv** con este formato:

```
item_id,description,stock
1001,Resistencia 10 kΩ,250
1002,Capacitor 100 μF,80
1003,Diodo 1N4148,500
...
```

y de un gran archivo **batch.csv** con 1 000 transacciones:

```
item_id,change
1002,-5
1001,10
```

1003,-20

...

[1.0 pts] Parte A: update_inventory.py (modo interactivo)

1. Al arrancar pide al usuario, en bucle, pares de valores:

ITEM_ID: <enter>

CHANGE: <enter>

<change> positivo suma stock, negativo lo resta.

2. Tras cada cambio:

- Actualiza en memoria el stock de `inventory.csv`.
- Añade una línea a `transactions.csv`:
timestamp,item_id,change,new_stock
2025-04-24T14:23:05,1002,-5,75
Usa `datetime.datetime.now().isoformat()`.
- Guarda inmediatamente el nuevo stock en `inventory.csv` (sobre-escribiendo toda la tabla).

3. Termina cuando el usuario ingresa `ITEM_ID` vacío.

Salida esperada

```
joant@LAPTOP-KAHN9ACA:~/lab4/preg3$ python3 update_inventory.py
ITEM_ID: 1001
CHANGE: -58
→ OK. Nuevo stock: 312
ITEM_ID: 1005
CHANGE: 42
→ OK. Nuevo stock: 437
ITEM_ID:
Fin modo interactivo.
```

[2.0 pts] Parte B: batch_update_sync.py (bloqueante)

1. Lee secuencialmente `batch.csv` con `csv.reader()`.
2. Para cada transacción:
 - Actualiza el stock en un dict cargado de `inventory.csv`.
 - Escribe inmediatamente (append) la línea correspondiente en `transactions.csv`.
 - Al terminar, sobre-escribe `inventory.csv` con los stocks finales.
 - Mide el tiempo total de ejecución con `time.perf_counter()` e imprime:
`Procesadas X transacciones en TTTT s`
 - Crear un archivo `T_sync.txt`, funcionalizar el código anterior y apilar en un bucle for al menos 10 líneas del tiempo total de ejecución.

Salida esperada

```
joant@LAPTOP-KAHN9ACA:~/lab4/preg3$ python3 batch_update_sync.py
Procesadas 1000 transacciones en 0.0061 s
```

[2.0 pts] Parte C: batch_update_sync.py (bloqueante)

1. Usa *aiofiles* para leer en paralelo el fichero batch.csv (p.ej. *asyncio.to_thread()* o *aiofiles.open()*).
2. Paraleliza el procesamiento de las 1 000 transacciones con *asyncio.gather()*:
 - Cada transacción actualiza el stock en el mismo dict (uso de *asyncio.Lock()* si es necesario).
 - Agrupa las escrituras de transactions.csv por bloques de 50 y haz un flush concurrente (ej. lanzando varias tareas de escritura).
3. Al terminar:
 - Guarda inventory.csv actualizado.
 - Imprime:
Procesadas X transacciones en TTTT s
 - Crear un archivo T_async.txt, funcionalizar el código anterior y apilar en un bucle for al menos 10 líneas del tiempo total de ejecución.

Salida esperada

```
● joant@LAPTOP-KAHN9ACA:~/lab4/preg3$ python3 batch_update_async.py
Procesadas 1000 transacciones en 0.0107 s
```

[2.0 pts] Parte D: speedup_test.py (evaluación de speed-up)

1. Leer cada archivo del reporte de tiempo de ejecución
2. Obtener el promedio para la version síncrona y asíncrona
3. Calcula el speed-up: T_{sync}/T_{async}
4. Imprime

Speed-up: S.Sx

Salida esperada

```
● joant@LAPTOP-KAHN9ACA:~/lab4/preg3$ python3 speedup_test.py
Avg sync: 0.0065 s
Avg async: 0.0107 s
Speed-up: 0.61x
```

Entregables

- Los scripts (update_inventory.py, batch_update_sync.py, batch_update_async.py, speedup_test.py) debidamente comentados.
- Los archivos inventory.csv, transactions.csv, T_sync.txt, T_async.txt