Análisis Lineal del Código: `reporting_logging.py`

El archivo `reporting_logging.py` es el responsable de toda la "memoria" a corto y largo plazo del sistema. Sus funciones principales son registrar cada evento de acceso (tanto exitoso como fallido), mantener contadores diarios, gestionar bloqueos de seguridad y generar reportes finales para auditoría. Es el historiador del sistema.

El script comienza importando las librerías necesarias: `json` para manejar archivos de estado en formato JSON, `csv` para generar reportes tabulares, `os` para interactuar con el sistema de archivos (crear carpetas, verificar si existen archivos), y `datetime` junto con `time` para manejar fechas y marcas de tiempo. También importa `messagebox` de Tkinter para poder mostrar notificaciones al usuario si se genera un reporte manualmente.

Se definen varias variables globales a nivel de módulo que almacenan el estado operativo del día. `contador_accesos_hoy` lleva la cuenta de cuántos accesos exitosos han ocurrido. `eventos_acceso_hoy` e `intentos_fallidos_hoy` son listas que almacenan diccionarios detallados de cada evento. `fecha_actual_para_conteo` guarda la fecha del día en curso para saber cuándo se debe resetear todo. Los diccionarios `intentos_fallidos_por_uid` y `accesos_recientes_uid` son cruciales para implementar las reglas de seguridad de bloqueo por insistencia y "anti-passback".

Una función clave es `cargar_estado_diario()`. Al iniciar el programa, esta función intenta leer un archivo llamado `estado_diario.json` (el nombre se obtiene de `constants.py`). Este archivo guarda el estado del último día de operación. Si la fecha guardada en el archivo coincide con la fecha actual, la función carga todos los contadores y logs, permitiendo que el programa se reanude como si nunca se hubiera cerrado. Si la fecha es de un día anterior, genera un reporte final para ese día que terminó, y luego resetea todas las variables a cero para comenzar un nuevo día. Esto asegura que los datos no se pierdan si la aplicación se reinicia. La función `guardar_estado_diario()` hace lo contrario: escribe el estado actual de todas estas variables en el archivo JSON, típicamente después de cada evento.

La función `verificar_y_resetear_por_cambio_de_dia()` se llama periódicamente (normalmente al inicio de cada evento) para comprobar si ha pasado la medianoche. Si la fecha actual es diferente a la almacenada, activa la lógica de cierre del día: genera el reporte final y reinicia los contadores.

La función `registrar_evento_acceso_exitoso(usuario_info)` se llama desde la máquina de estados cada vez que se concede un acceso. Incrementa el contador, crea un diccionario con todos los detalles del evento (quién, cuándo, con qué credencial) y lo añade a la lista `eventos_acceso_hoy`. Además, actualiza el diccionario `accesos_recientes_uid` para iniciar el "cooldown" de esa credencial y resetea el contador de fallos para ese usuario, si lo tuviera.

Por otro lado, `registrar_intento_fallido(...)` se llama cuando un acceso es denegado. Registra los detalles del intento fallido y, si el parámetro `contar_para_bloqueo_insistencia` es `True`, actualiza el contador de fallos para esa credencial. Si el número de fallos alcanza el umbral definido en `constants.MAX_INTENTOS_FALLIDOS_UID`, la función calcula un tiempo de bloqueo (usando los niveles de `constants.TIEMPO_BLOQUEO_UID_NIVEL`) y actualiza el diccionario `intentos_fallidos_por_uid`, marcando hasta qué hora esa credencial estará inactiva.

Finalmente, `generar_reporte_final_dia(...)` toma todos los datos recopilados durante un día y los exporta a formatos permanentes y legibles por humanos. Crea una carpeta (definida en `constants.CARPETA_REPORTES`) si no existe. Luego, genera un archivo JSON completo con todos los logs y un resumen, y también crea dos archivos CSV separados: uno para los accesos exitosos y otro para los fallidos, ideales para ser abiertos en programas como Excel para un análisis más detallado.

El bloque `if __name__ == '__main__':` permite probar este módulo de forma aislada. Simula una serie de accesos exitosos y fallidos, prueba la lógica de bloqueo y cooldown, y finalmente genera un reporte de prueba, lo que facilita la verificación de toda la lógica de registro y reporte sin necesidad de interactuar con el hardware o la aplicación completa.