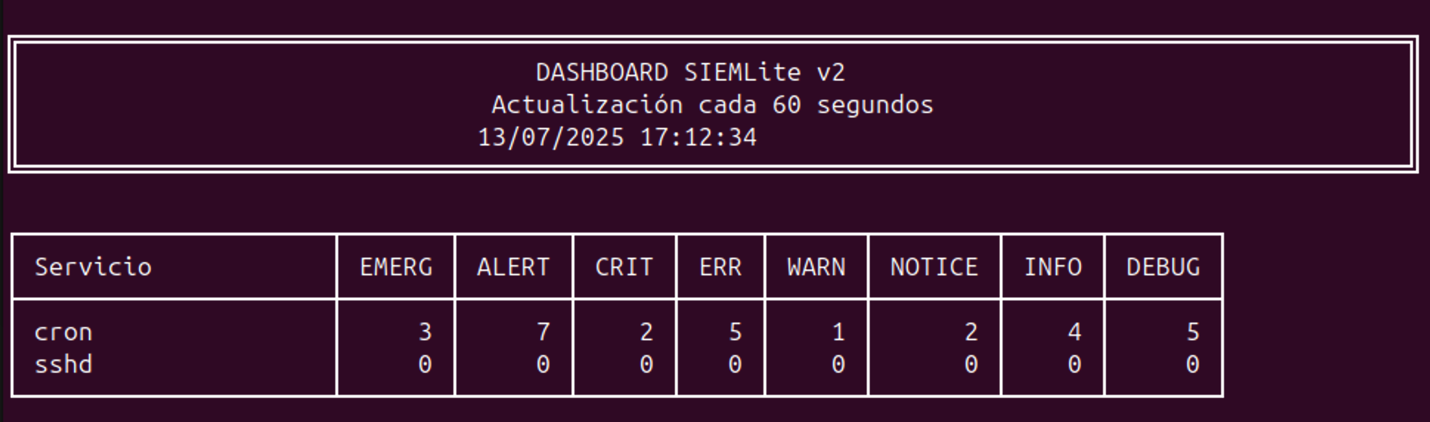
A close up of a logo

Description automatically generated

**ESCUELA SUPERIOR POLICTENCIA DEL LITORAL**

**FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN**

**SISTEMAS OPERATIVOS**

**PROYECTO PARCIAL SIEMLITE**

**ANGELO SAUL ZURITA GUERRERO**

**202208443**

# Objetivo

Implementar una solución de monitoreo en tiempo real de los archivos logs de un sistema GNU/Linux.

# Requerimientos funcionales

* Parámetros configurables:
  + Permitir especificar los nombres de los servicios a monitorear (mínimo 2).
  + Definir el tiempo de actualización de la información (como parámetro o constante).
* Manejo de niveles de prioridad de los mensajes: alert, err, notice, info, debug, etc.
* Generar un dashboard en modo texto que:
  + Muestre la cantidad de mensajes generados.
  + Se actualice conforme al tiempo especificado.
* Lectura de logs:
  + Usar el comando journalctl para leer mensajes según prioridad y servicio.
  + Implementar esto mediante una llamada del sistema exec.
* Sistema de alertas:
  + Al superar un threshold de mensajes, emitir una alerta vía mensaje de WhatsApp (sugerencia: usar la API de Twilio).
* Deberá implementar un programa en C que genere una anomalía en el funcionamiento de uno de los servicios que se encuentra monitoreando, puede ser por estrés o fallos en las peticiones, etc. donde se pueda evidenciar la actualización de los archivos log para su posterior notificación vía mensaje.

# Codificación

## Manejo de Parámetros de Entrada



Desarrollé un sistema de procesamiento de argumentos usando la biblioteca estándar getopt() que valida servicios (mínimo 2) y tiempo de actualización configurable. Utilicé doble validación, primero con la propia de getopt(), luego con strtol() para conversión segura de números, y finalmente validación de rangos específicos. Creé la función validar\_servicio() que implementa una expresión regular implícita usando isalnum() para verificar caracteres válidos en nombres de servicios.

**Funciones clave utilizadas:**

* getopt() para parsing estándar POSIX de argumentos
* strtol() con verificación de errno para conversión numérica robusta
* Validación personalizada de caracteres alfanuméricos y símbolos permitidos

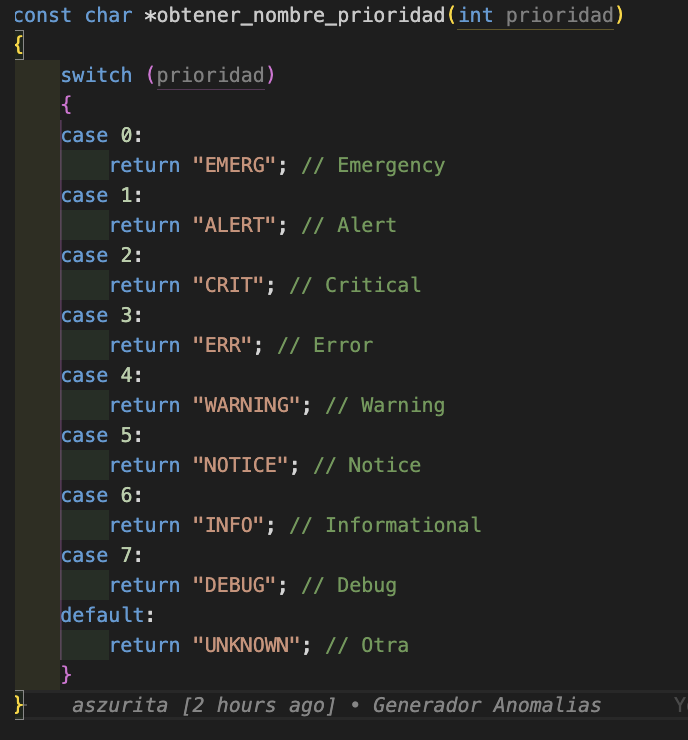
## Uso de journalctl y Llamadas al Sistema

Desarrollé una arquitectura multi-proceso donde cada servicio es monitoreado por un proceso hijo independiente que ejecuta journalctl usando execlp(). Use métodos como fork/exec creando la función ejecutar\_journalctl(). Utilicé redirección de descriptores de archivo con dup2() para capturar el output de journalctl y enviarlo través de pipes al proceso padre.

**Funciones clave utilizadas:**

* fork() para crear procesos hijos paralelos
* execlp() para ejecutar journalctl con filtros específicos
* dup2() para redireccionar stdout/stderr al pipe
* waitpid() para sincronización y prevención de procesos zombie

## Manejo de Niveles de Prioridad



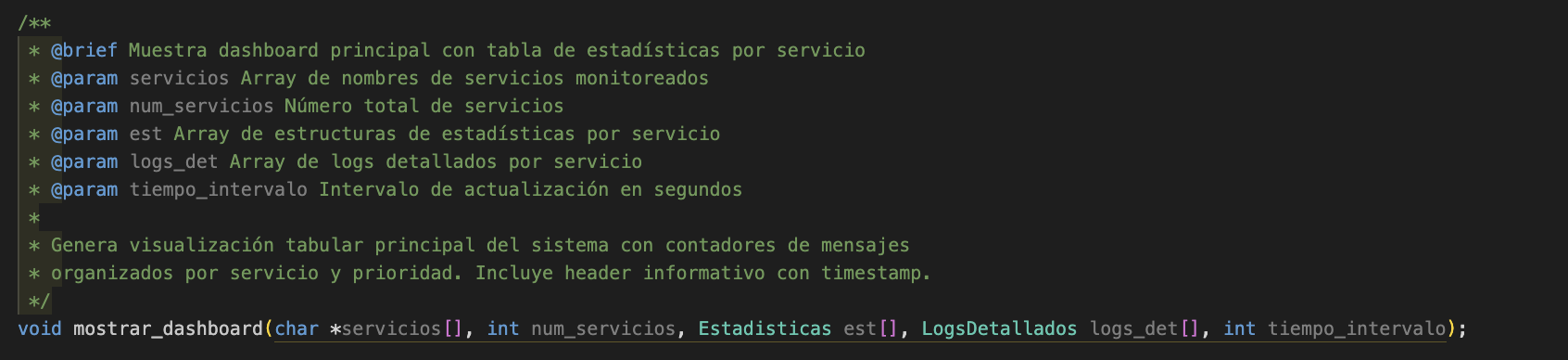
Implementé clasificación completa de los 8 niveles de prioridad syslog , desde emergency (0) hasta debug (7).

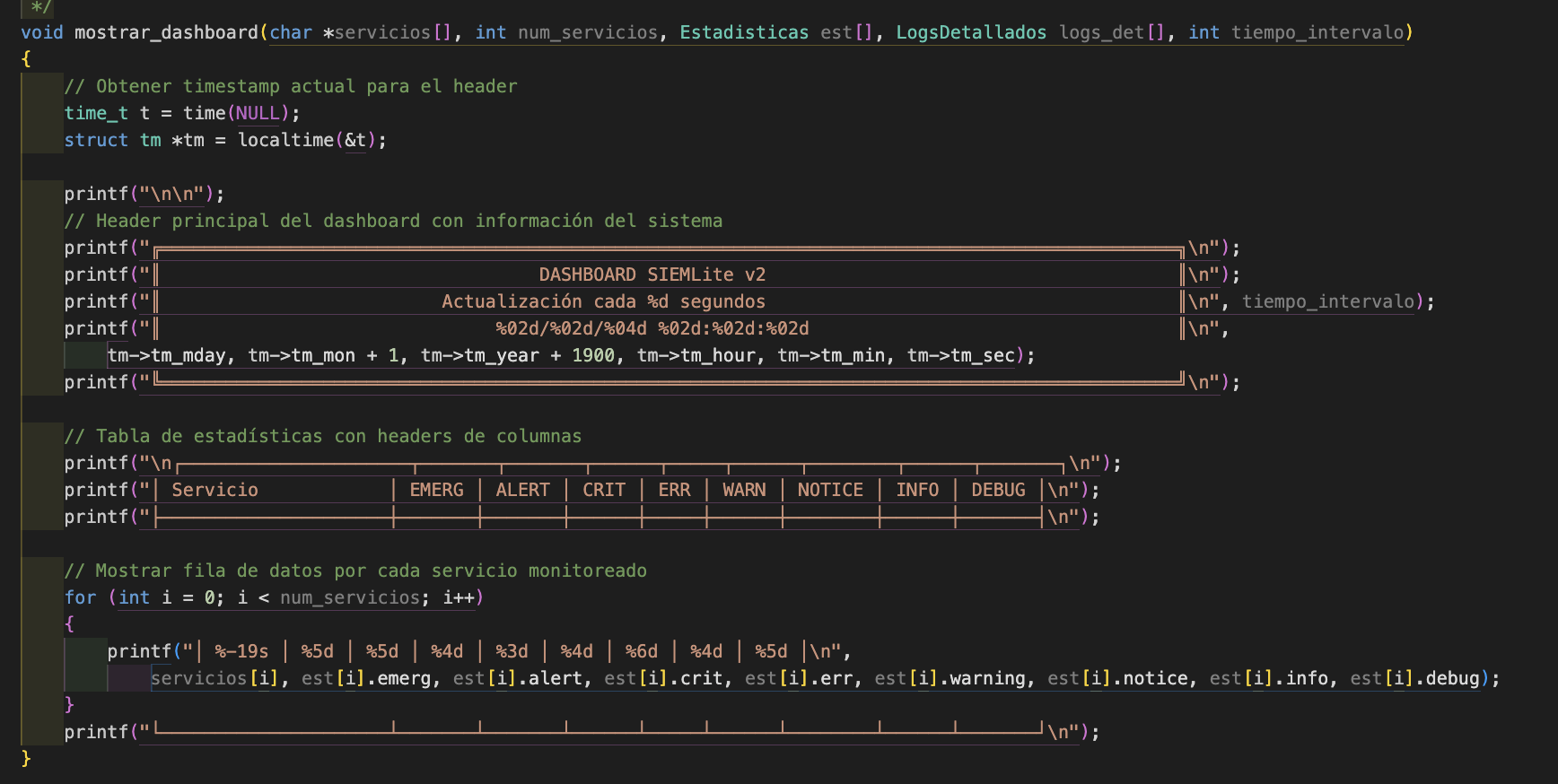
Diseñé una estructura de datos Estadisticas que mapea cada nivel de prioridad a un contador específico. Desarrollé la función analizar\_logs\_completos() que parsea el formato export de journalctl buscando el campo PRIORITY= y clasifica cada mensaje.

**Funciones clave utilizadas:**

* strncmp() para parsing eficiente de campos del formato export
* atoi() para conversión de prioridad string a numérico
* Switch statements para clasificación rápida por prioridad
* Estructura typedef personalizada para organización de datos

## Dashboard en Modo Texto



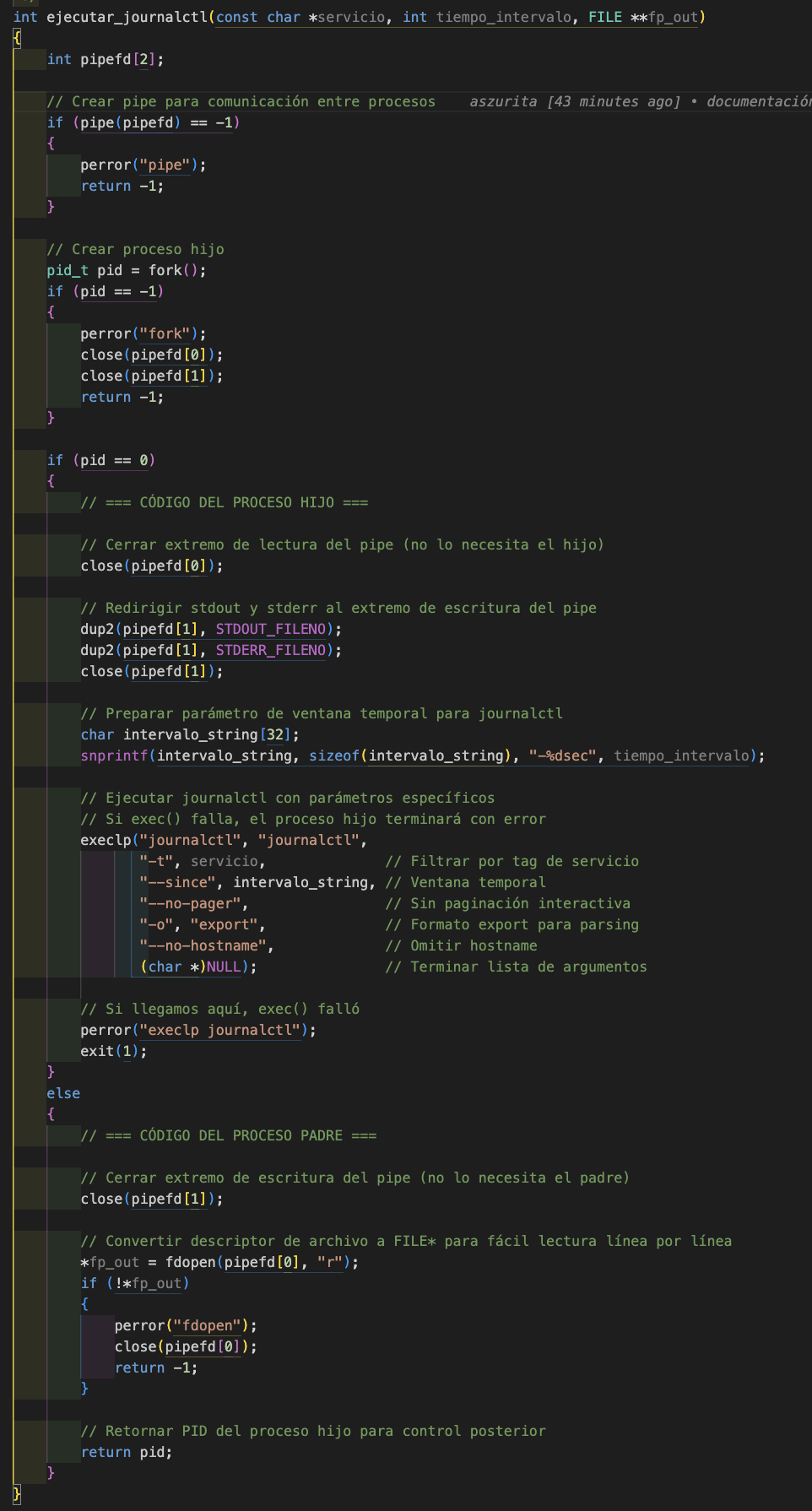


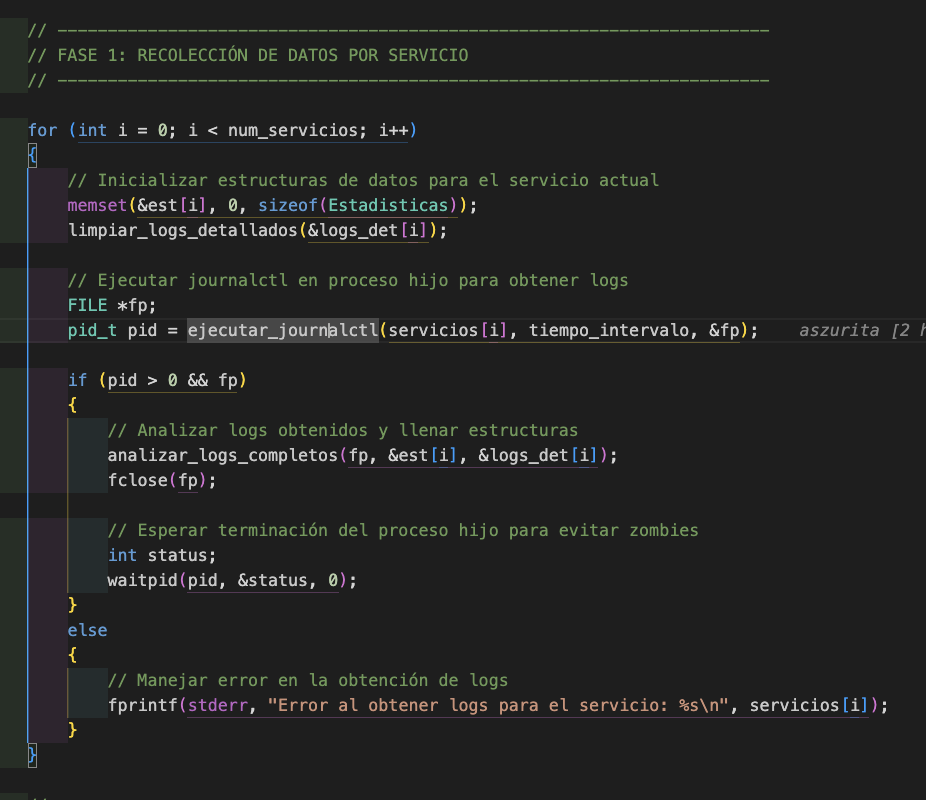
Creé un dashboard visual usando caracteres que muestra estadísticas en tiempo real organizadas en tabla por servicio y prioridad. Desarrollé la función mostrar\_dashboard() que utiliza caracteres Unicode para crear bordes visuales profesionales. Implementé formateo de timestamps con strftime() y alineación de columnas usando especificadores de formato printf() con padding.

**Funciones clave utilizadas:**

* time() y localtime() para timestamps en tiempo real
* strftime() para formateo de fecha/hora legible
* printf() con especificadores de ancho para alineación tabular
* Caracteres para presentación visual profesional

## Procesos Hijos para Monitoreo



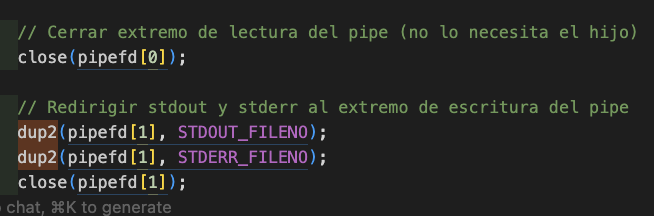


Implementé una arquitectura de procesos donde el proceso padre coordina y cada proceso hijo maneja un servicio específico ejecutando journalctl. Utilicé el patrón de diseño Producer-Consumer donde cada proceso hijo actúa como productor de datos (ejecutando journalctl) y el proceso padre como consumidor (analizando logs). Implementé manejo de ciclo de vida completo con waitpid() para evitar procesos zombie.

**Funciones clave utilizadas:**

* fork() para creación de procesos hijos
* exec() family para reemplazo de imagen de proceso
* waitpid() con flags específicos para manejo de estados
* Control de descriptores de archivo para comunicación

## Comunicación entre Procesos



Desarrollé un sistema de comunicación unidireccional usando pipes anónimos para transferir datos de logs desde procesos hijos al proceso padre. Implementé el patrón IPC (Inter-Process Communication) usando pipe() para crear canales de comunicación. Utilicé fdopen() para convertir descriptores de archivo a streams FILE\* facilitando lectura línea por línea. Implementé sincronización explícita cerrando descriptores apropiados en cada proceso.

**Funciones clave utilizadas:**

* pipe() para crear canales de comunicación bidireccional
* fdopen() para conversión de descriptores a streams
* close() para manejo explícito de descriptores
* fgets() para lectura línea por línea desde pipes

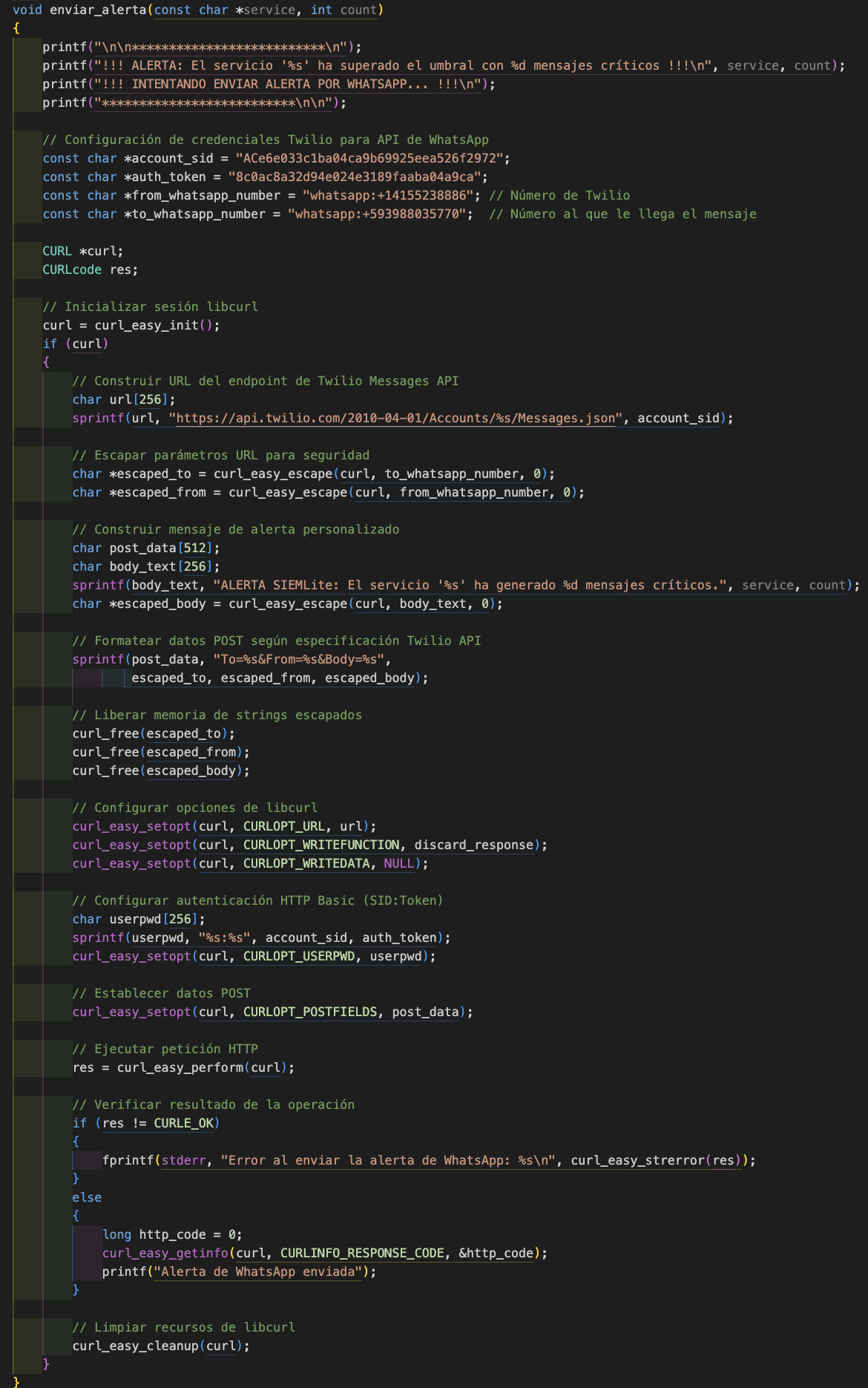
## Sistema de Alertas y Threshold

Desarrollé un sistema de alertas configurables que monitorea umbrales por servicio y dispara notificaciones automáticas cuando se superan límites establecidos.. Implementé la función verificar\_alertas() que aplica lógica de análisis threshold calculando totales de mensajes por servicio. Utilicé parámetros configurables vía línea de comandos para personalizar sensibilidad del sistema. Integré llamadas automáticas al sistema de notificaciones.

**Funciones clave utilizadas:**

* Aritmética de acumulación para cálculo de totales
* Lógica condicional para comparación contra thresholds
* Invocación automática de enviar\_alerta() cuando se superan límites
* Configuración persistente de umbrales via variables globales

## Integración con API de Notificaciones

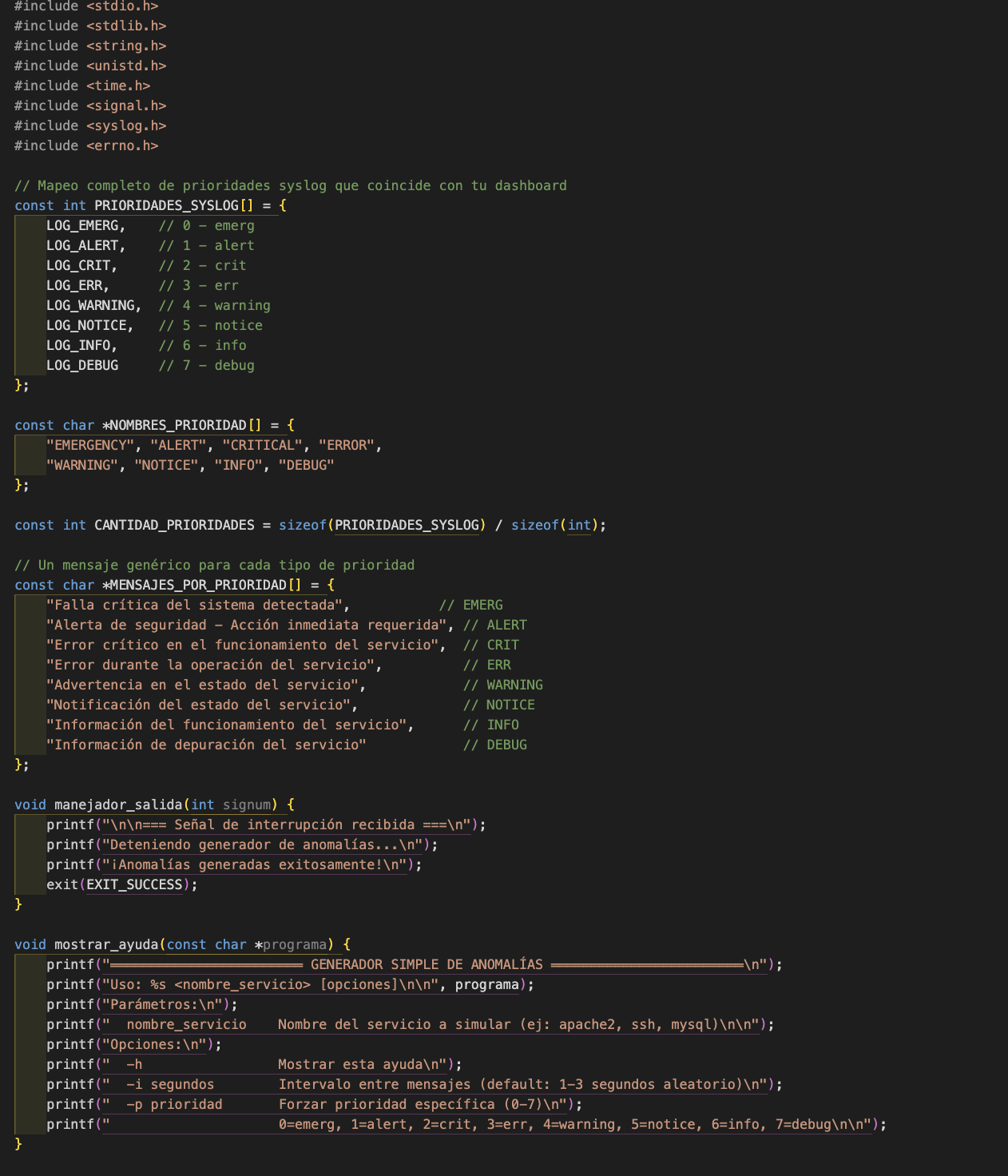


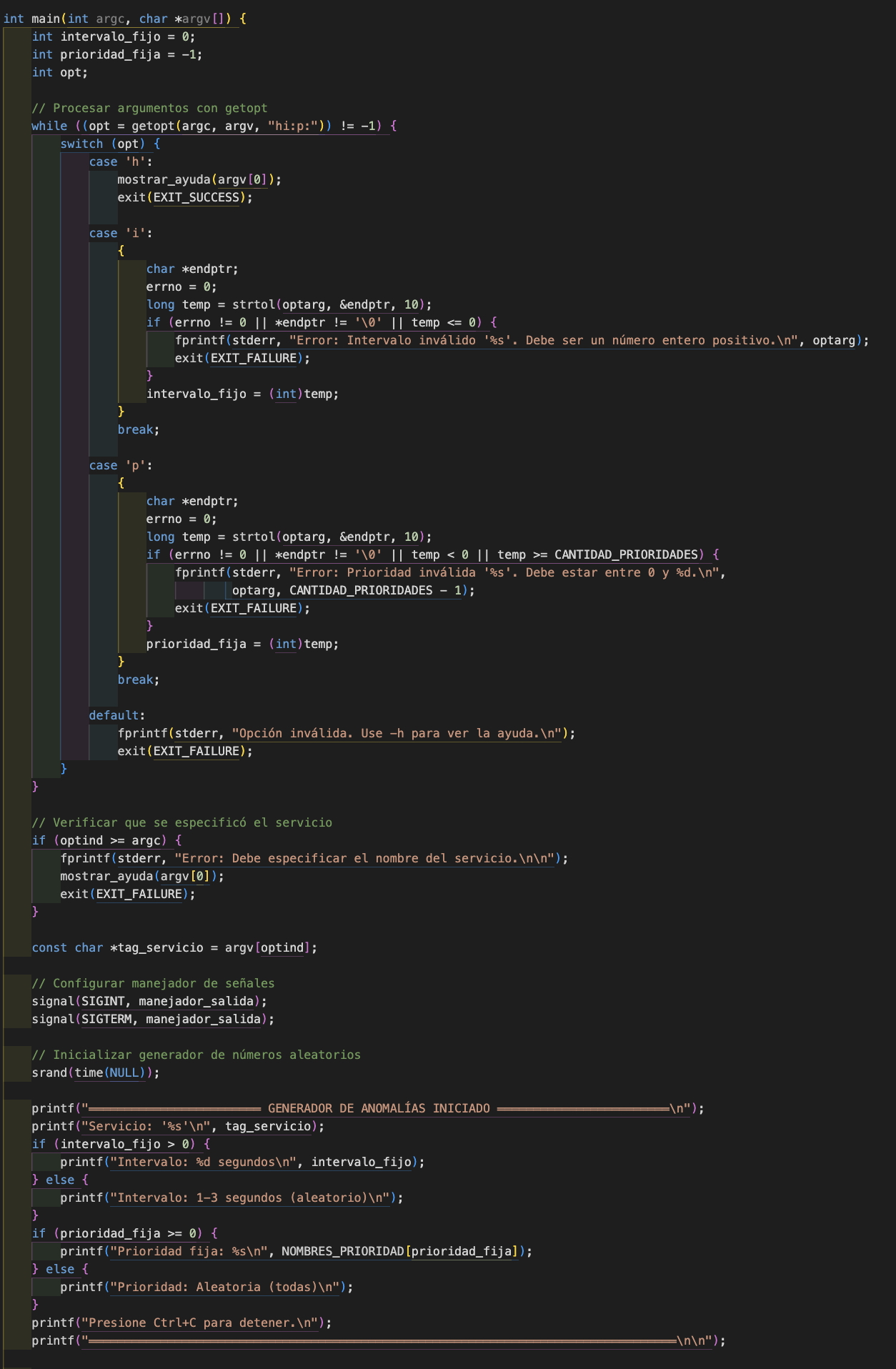
Integré completamente la API de Twilio para envío de alertas WhatsApp usando libcurl para peticiones HTTP autenticadas. Desarrollé la función enviar\_alerta() que implementa el protocolo REST de Twilio. Utilicé libcurl para manejo de peticiones HTTP POST con autenticación Basic. Implementé escape de parámetros URL y construcción dinámica de payloads JSON.

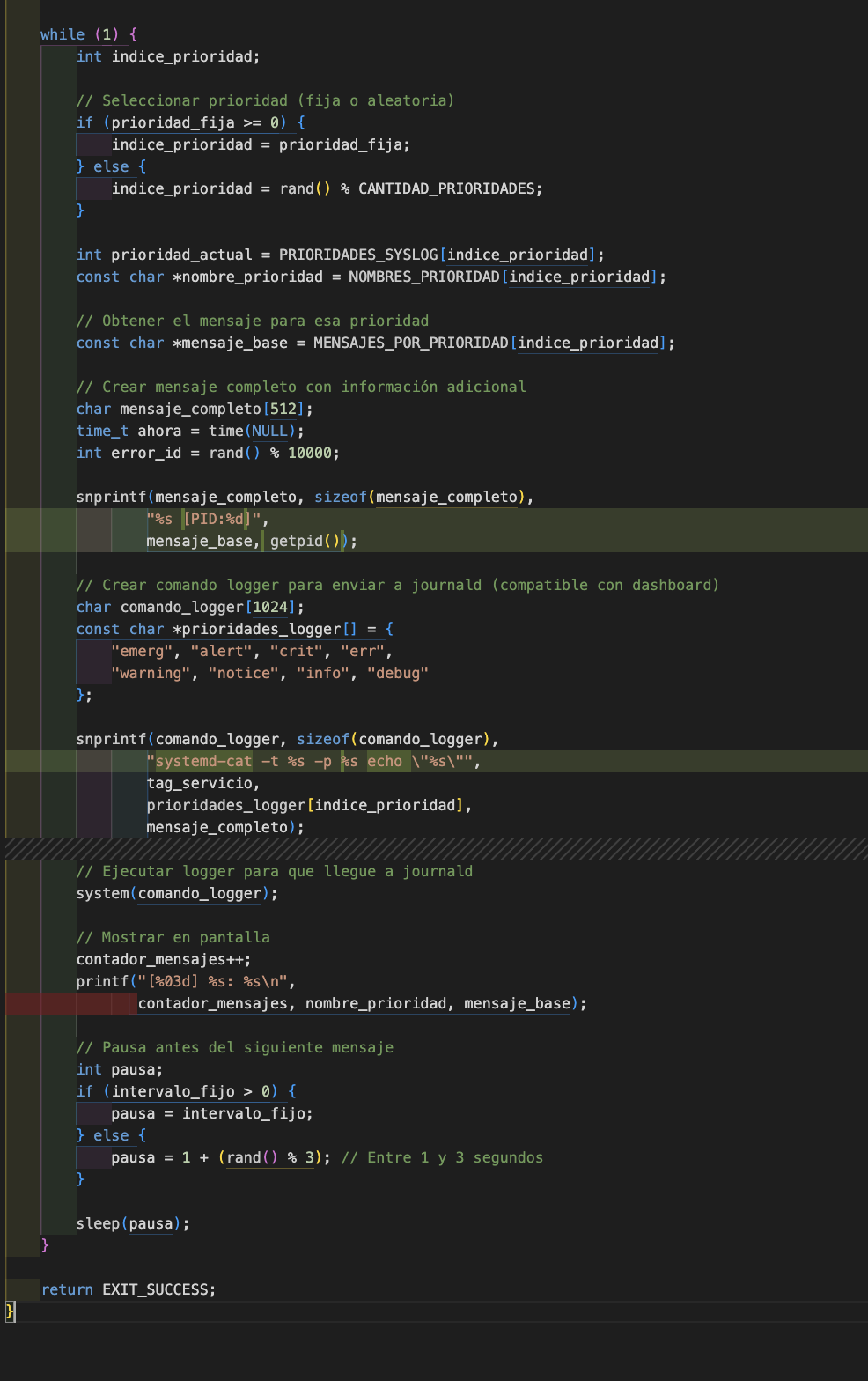
**Funciones clave utilizadas:**

* curl\_easy\_init() para inicialización de sesiones HTTP
* curl\_easy\_setopt() para configuración de headers y autenticación
* curl\_easy\_escape() para sanitización de parámetros URL
* curl\_easy\_perform() para ejecución de peticiones
* Manejo de códigos de respuesta HTTP para verificación

## Implementación de Generador de Anomalía en Servicio







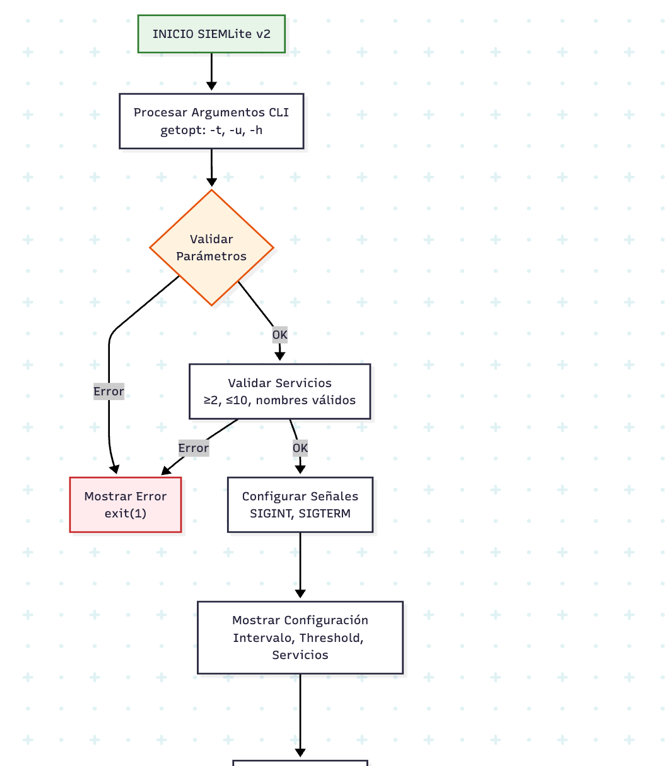
Desarrollé una herramienta independiente de testing que simula comportamientos anómalos generando logs controlados para validar el sistema de monitoreo. Creé el programa generarAnomalia.c que utiliza systemd-cat para envío directo a journald. Implementé generación aleatoria de prioridades y mensajes usando rand() con semilla temporal. Desarrollé interfaz de línea de comandos con opciones para controlar frecuencia y tipos de eventos.

**Funciones clave utilizadas:**

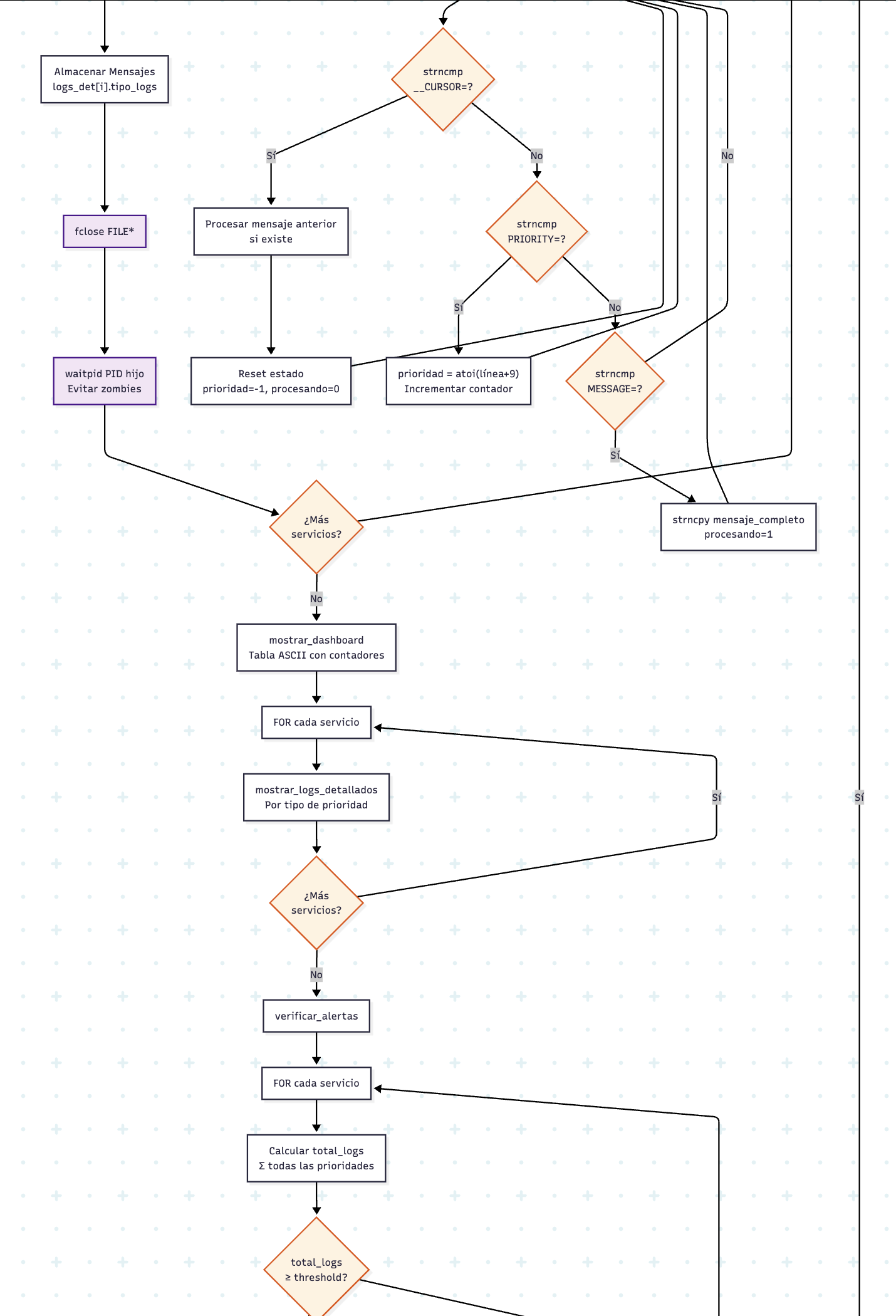
* srand() con time() para inicialización de generador aleatorio
* rand() para selección de prioridades y intervalos
* system() para ejecución de comandos systemd-cat
* getopt() para procesamiento de argumentos de configuración
* signal() para manejo elegante de terminación

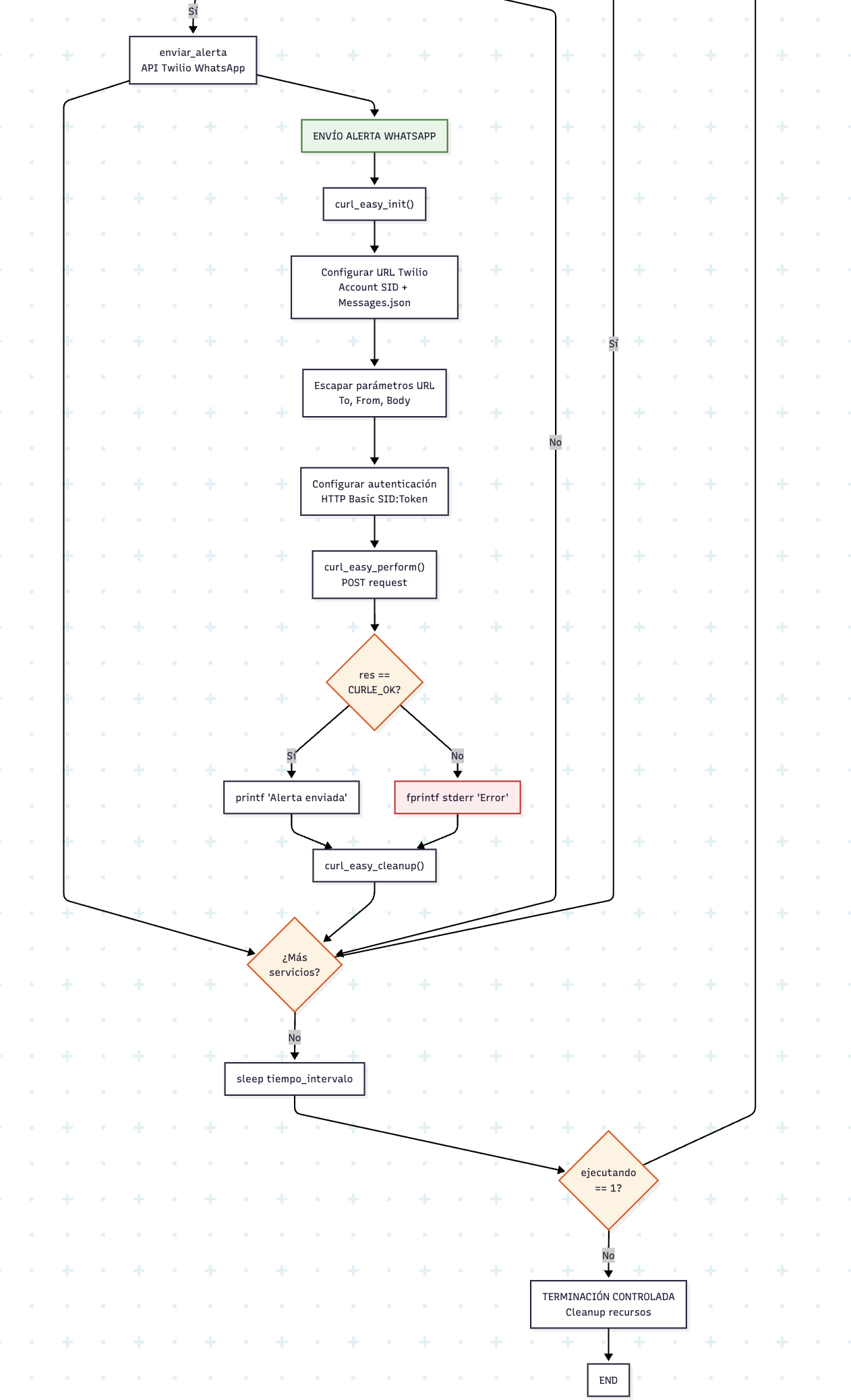
# Diagrama

Se realizo un diagrama a nivel de software, donde se pueda ver el flujo del algoritmo completo, independientemente de la modularización del código.



# 





# 

# Demo

<https://drive.google.com/drive/folders/1vh5nf3nKn2LDr8zM3Zu9_LJVt1hOTr3k?usp=sharing>

# Uso de inteligencia artificial

Porcentaje 10%

# Referencias

Freedesktop.org. (s. f.). systemd‑journald.service. En systemd-manpages. [https://www.freedesktop.org/software/systemd/man/latest/systemd-journald.service.html](https://www.freedesktop.org/software/systemd/man/latest/systemd-journald.service.html" \t "_new)

**Twilio. (s. f.).** Programmable Messaging for WhatsApp and C#/.NET Quickstart. <https://www.twilio.com/docs/whatsapp/quickstart/csharp>

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating system concepts (10.ª ed.). Wiley.

**Doxygen. (s. f.).** Documenting the code: Special comment blocks (C/C++). <https://doxygen.nl/manual/docblocks.html#cppblock>