# Trabajo Práctico Especial: Proxy SOCKSv5

**Grupo 20**

Nicolás Valentín Arias (62272)
Tomás Pinausig (63167)
Javier Peral (62289)
Santiago Mesa Rubio (63456)

# Índice

# 1. Descripción detallada de protocolos y aplicaciones

## 1.1. Servidor Proxy SOCKSv5

Se implementó un servidor proxy que cumple con el RFC 1928. El servidor maneja múltiples clientes de forma concurrente utilizando un único hilo de ejecución (single-threaded) mediante E/S no bloqueante y multiplexación con `select/poll`. El flujo de conexión implementado es:

1. **Handshake (HELLO):** Negociación de versión y método de autenticación.
2. **Autenticación (AUTH):** Soporte para RFC 1929 (Usuario/Contraseña).
3. **Solicitud (REQUEST):** Soporte para comando CONNECT.
4. **Transmisión (COPY):** Túnel de datos bidireccional entre cliente y servidor de origen.

## 1.2. Protocolo de Monitoreo (MCP)

Se diseñó un protocolo binario a medida (Monitoring  Configuration Protocol - MCP) que corre en un puerto separado (default 8080).

- **Arquitectura:** Cliente-Servidor sobre TCP.
- **Funcionalidad:** Permite obtener métricas en tiempo real, listar usuarios y modificar la configuración de seguridad (agregar/borrar usuarios) sin reiniciar el proxy.
- **Formato:** Header fijo de 4 bytes (Ver, Cmd, Len) + Payload variable.

# 2. Problemas encontrados durante el diseño e implementación

## 2.1. Lecturas Parciales

Uno de los mayores desafíos fue manejar la fragmentación de paquetes TCP. Al usar sockets no bloqueantes, las llamadas a `recv` a menudo retornaban menos bytes de los esperados para una estructura completa (ej: header SOCKS). **Solución:** Se implementaron parsers con estado interno que pueden "pausar.el procesamiento y reanudarlo cuando llegan más datos, retornando un estado `INCOMPLETE`.

## 2.2. Resolución DNS Bloqueante

La función `getaddrinfo` es bloqueante por estándar POSIX. Su uso directo detenía todo el servidor. **Solución:** Se implementó un pool de hilos dinámico para resolver nombres de dominio. El hilo principal delega la tarea y el hilo trabajador notifica la finalización mediante `selector_notify_block`.

# 3. Limitaciones de la aplicación

- **Comandos SOCKS:** Solo se soporta el comando `CONNECT`. Los comandos `BIND` y `UDP ASSOCIATE` no están implementados.
- **Escalabilidad Vertical:** Al ser single-threaded, el servidor no aprovecha múltiples núcleos de CPU para el procesamiento de paquetes, aunque es muy eficiente en I/O.
- **Buffer Fijo:** El tamaño del buffer de lectura/escritura es fijo en tiempo de compilación (4KB), lo que podría no ser óptimo para todas las redes.

# 4. Posibles extensiones

- Implementar soporte para IPv6 completo en el protocolo de monitoreo.
- Agregar soporte para el comando `UDP ASSOCIATE` para permitir tráfico UDP.
- Implementar un pool de buffers de memoria para reducir las llamadas a `malloc/free`.
- Añadir soporte para SSL/TLS en el puerto de monitoreo para seguridad.

# 5. Conclusiones

El desarrollo permitió consolidar los conocimientos sobre programación de redes en C y el manejo de concurrencia sin paralelismo. La arquitectura basada en selectores demostró ser robusta y eficiente para manejar cientos de conexiones simultáneas con bajo consumo de recursos. La modularización del código facilitó la implementación de protocolos adicionales como el de monitoreo.

# 6. Ejemplos de prueba

## 6.1. Prueba básica con cURL

Comando para probar la conexión a través del proxy:

```
curl -x socks5://juan:1234@localhost:1080 http://www.google.com
```

Resultado esperado: El HTML de la página de Google.

## 6.2. Prueba de Monitoreo

Uso del cliente para ver métricas:

```
./bin/socks5_client -u admin:1234 metrics
```

Salida:

```
Server Metrics:
  Historical connections: 15
  Current connections:    2
  Bytes transferred:      45023
```

# 7. Guía de instalación detallada

## 7.1. Requisitos

- Sistema Operativo: Linux (probado en Ubuntu 20.04+) o POSIX compatible.
- Compilador: GCC o Clang con soporte C11.
- Herramientas: Make.

## 7.2. Compilación

Desde la raíz del proyecto, ejecutar:

```
make clean
make all
```

Esto generará los binarios en el directorio `bin/`: `socks5d` (servidor) y `socks5_client` (cliente de monitoreo).

# 8. Instrucciones para la configuración

El servidor se configura principalmente mediante argumentos de línea de comandos. No requiere archivos de configuración externos.

Argumentos principales:

- `-p <puerto>`: Puerto de escucha SOCKS (def: 1080).
- `-P <puerto>`: Puerto de monitoreo (def: 8080).
- `-u <user:pass>`: Define un usuario. Puede usarse hasta 10 veces.
- `-o <archivo>`: Ruta al archivo de log de accesos.

# 9. Ejemplos de configuración y monitoreo

## 9.1. Iniciar servidor seguro

Escuchar solo en localhost, con log y usuario admin:

```
./bin/socks5d -l 127.0.0.1 -p 8888 -u admin:secreto -o access.log
```

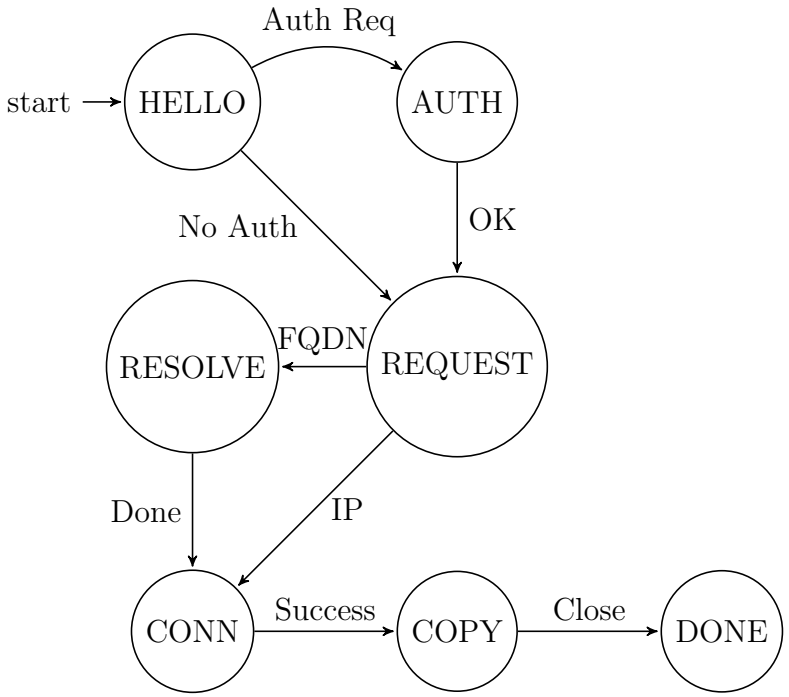
## 9.2. Agregar usuario en tiempo de ejecución

Desde otra terminal:

```
./bin/socks5_client -P 8080 -u admin:secreto adduser nuevo:123
```

# 10. Documento de diseño del proyecto

## 10.1. Diagrama de Estados del Protocolo (FSM)




## 10.2. Módulos Principales

- **selector.c:** Wrapper de `select/pselect` para manejo de eventos.
- **stm.c:** Motor genérico de máquinas de estado.
- **socks5nio.c:** Lógica específica del protocolo SOCKSv5. Define los estados y transiciones.
- **buffer.c:** Manejo seguro de memoria para I/O.
- **monitoring.c:** Implementación del servidor de administración.