Informe Trabajo Práctico Especial: Servidor Proxy SOCKSv5

Materia: 72.07 - Protocolos de Comunicación
Cuatrimestre: 2025/2
Grupo: [Completar Número]
Integrantes:

- [Nombre Completo] - [Legajo]
- [Nombre Completo] - [Legajo]
- [Nombre Completo] - [Legajo]

---

1. Índice

---

2. Descripción detallada de los protocolos y aplicaciones desarrolladas

El proyecto consiste en el desarrollo de un servidor proxy que implementa el protocolo **SOCKS versión 5 (RFC 1928)**, junto con un protocolo de gestión propietario para la administración y monitoreo en tiempo real.

Servidor SOCKSv5 (`socks5d`)

La aplicación principal es un servidor concurrente basado en una arquitectura de **Event Loop** (bucle de eventos) utilizando E/S no bloqueante. Se utilizó la librería `selector.c` provista por la cátedra para abstraer la llamada al sistema `select()` .

Funcionalidades implementadas:

- **Negociación:** Soporte para métodos de autenticación `NO AUTHENTICATION REQUIRED` (0x00) y `USERNAME/PASSWORD` (0x02).
- **Autenticación (RFC 1929):** Validación de usuarios contra un registro en memoria.
- **Comando CONNECT:** Establecimiento de conexiones TCP hacia destinos especificados por IPv4, IPv6 o Nombre de Dominio (FQDN).
- **Streaming de Datos:** Transferencia bidireccional de datos entre el cliente y el servidor de origen ("pipe") utilizando buffers dinámicos.
- **Resolución DNS Asíncrona:** Para evitar el bloqueo del servidor durante la resolución de nombres (FQDN), se implementó un modelo híbrido donde la llamada `getaddrinfo` se ejecuta en un hilo separado (*worker thread*), notificando al selector principal al finalizar.

Protocolo de Gestión (Monitor)

Se implementó un protocolo de texto sobre TCP (puerto por defecto 8080) que permite administrar el servidor sin reiniciarlo. El protocolo sigue un esquema solicitud-respuesta similar a POP3.

Comandos del protocolo:

- `AUTH <user> <pass>` : Autenticación de administrador.
- `STATS` : Consulta de métricas (conexiones históricas, actuales, bytes transferidos).
- `USERS` : Listado de usuarios proxy activos.
- `ADDUSER <user> <pass>` : Alta de usuarios en tiempo de ejecución.
- `DELUSER <user>` : Baja de usuarios en tiempo de ejecución.

Cliente de Gestión (`client`)

Se desarrolló una herramienta de línea de comandos que implementa el protocolo de gestión, permitiendo a los administradores enviar comandos y visualizar las respuestas de forma amigable.

---

3. Problemas encontrados durante el diseño y la implementación

1. **Bloqueo por Resolución DNS:** La función `getaddrinfo` es bloqueante. En un diseño *single-threaded*, una demora en el DNS congelaría a todos los clientes conectados.
    - **Solución:** Se diseñó el estado `REQUEST_RESOLVING` en la máquina de estados. Al entrar en este estado, se lanza un hilo POSIX (`pthread`) detached que realiza la resolución. El file descriptor del cliente se desregistra temporalmente de eventos de escritura/lectura (`OP_NOOP`) y, al finalizar el hilo, se utiliza `selector_notify_block` para despertar al bucle principal y retomar la conexión en el estado `REQUEST_CONNECTING` .

2. **Manejo de Buffers y `EAGAIN`:** Garantizar la integridad de los datos en E/S no bloqueante requiere manejar escrituras parciales.
    - **Solución:** Se utilizaron buffers circulares (`buffer.c`) para almacenar los datos pendientes. Si `send()` devuelve `EAGAIN` o escribe menos bytes de los solicitados, el sistema se suscribe a eventos de escritura (`OP_WRITE`) y reintenta cuando el socket está disponible.
3. **Límite de File Descriptors:** Durante las pruebas de carga, el servidor alcanzaba el límite de 1024 descriptores impuesto por `FD_SETSIZE` en `select()`.
    - **Solución:** Se optimizó la gestión de memoria y cierre de conexiones para evitar fugas (*leaks*), asegurando que los sockets se cierren y desregistren correctamente en el estado `DONE` o `ERROR`.

---

4. Limitaciones de la aplicación

1. **Comandos SOCKS:** Solo se soporta el comando `CONNECT`. Los comandos `BIND` y `UDP ASSOCIATE` no están implementados, limitando el uso para protocolos que requieren UDP o conexiones entrantes al cliente.
2. **Escalabilidad:** El uso de `select()` limita la cantidad máxima de conexiones simultáneas a aproximadamente 1024 (menos los FDs reservados). Para escalar a miles de conexiones (C10K), sería necesario migrar a `poll`, `epoll` o `kqueue`.
3. **Persistencia:** Los usuarios y métricas se almacenan en memoria volátil. Al reiniciar el proceso `socks5d`, esta información se pierde.

---

5. Posibles extensiones

1. **Persistencia:** Implementar una base de datos simple (archivo plano o SQLite) para guardar usuarios y logs de auditoría de forma persistente.
2. **Soporte IPv6 Completo:** Extender la capacidad del servidor de gestión para escuchar nativamente en direcciones IPv6 (actualmente prioriza IPv4 en el cliente).
3. **Sniffing de Tráfico:** Agregar un módulo "disector" para analizar el tráfico en texto claro (HTTP/POP3) y extraer credenciales, tal como se sugiere para una segunda etapa del proyecto.

---

6. Conclusiones

Se ha desarrollado un servidor proxy SOCKSv5 robusto que cumple con los requerimientos funcionales y de performance. La arquitectura basada en eventos demostró ser superior a los modelos tradicionales de "un hilo por cliente" en términos de consumo de memoria y *context switching*. Las pruebas de estrés validaron la capacidad del servidor para manejar cargas elevadas (hasta 1000 clientes concurrentes) manteniendo tiempos de respuesta bajos y estabilidad operativa.

---

7. Ejemplos de prueba

Las pruebas de estrés se realizaron utilizando scripts automatizados en Python en un entorno macOS (Apple Silicon).

1. Conexiones Simultáneas: El servidor mantuvo **1000 conexiones concurrentes** realizando el handshake completo sin errores.

- **Resultado:** 1000 Éxitos / 0 Fallos.
- **Tiempo total:** 0.26 segundos para establecer las conexiones.

2. Throughput: Se midió la capacidad de procesamiento de nuevas conexiones por segundo.

- **Throughput Sostenido:** ~486.8 conexiones/segundo.
- **Pico (Burst):** ~3,887 conexiones/segundo.

3. Latencia: El tiempo de respuesta promedio se mantuvo por debajo de los **3ms** incluso bajo carga máxima, demostrando la eficiencia del manejo no bloqueante.

---

8. Guía de instalación detallada y precisa

Requisitos

- Sistema operativo compatible con POSIX (Linux, macOS, BSD).
- Compilador `gcc` (soporte C11).
- Herramienta `make`.

Pasos

1. Descomprimir el código fuente o clonar el repositorio.
2. Desde la raíz del proyecto, ejecutar:

    ```
    make all
    ```

    Este comando compilará el servidor y el cliente, generando los ejecutables `socks5d` y `client` en el directorio actual (o `bin/` según configuración local).
3. Para eliminar los archivos generados:

```
make clean
```

---

9. Instrucciones para la configuración

El servidor se configura íntegramente a través de argumentos de línea de comandos al momento de la ejecución. No requiere archivos de configuración externos.

Ejecución

```
./socks5d [OPCIONES]
```

Opciones Principales

Las siguientes opciones están disponibles, siguiendo los lineamientos POSIX:

- `-h` : Imprime la ayuda y termina la ejecución.
- `-l <SOCKS addr>` : Dirección IP donde servirá el proxy SOCKS. Por defecto escucha en `0.0.0.0` (todas las interfaces).
- `-p <SOCKS port>` : Puerto TCP entrante para conexiones SOCKS. Por defecto es `1080`.
- `-L <conf addr>` : Dirección IP donde servirá el servicio de gestión (Management). Por defecto es `127.0.0.1` (loopback) por seguridad.
- `-P <conf port>` : Puerto TCP entrante para el protocolo de configuración/gestión. Por defecto es `8080`.
- `-u <name>:<pass>` : Registra un usuario inicial en el proxy con su contraseña. Esta opción puede repetirse hasta 10 veces para precargar múltiples usuarios.
  - *Ejemplo:* `-u admin:1234 -u user:pass`
- `-N` : Desactiva los disectores de credenciales (funcionalidad reservada para futuras extensiones de sniffing).
- `-v` : Imprime información sobre la versión del servidor y termina.

---

10. Ejemplos de configuración y monitoreo

Configuración del Servidor

Caso 1: Servidor de prueba local Ejecuta el servidor en los puertos por defecto, creando un usuario administrador.

```
./socks5d -u admin:admin123
```

Caso 2: Servidor público con gestión segura Ejecuta el proxy en todas las interfaces (0.0.0.0) puerto 1080, pero restringe la gestión a localhost puerto 9090.

```
./socks5d -l 0.0.0.0 -p 1080 -L 127.0.0.1 -P 9090 -u admin:secret
```

Monitoreo con Cliente de Gestión

Una vez iniciado el servidor, se utiliza el binario `client` para administrarlo.

Conexión:

```
./client -L 127.0.0.1 -P 8080 -u admin -p admin123
```

Ejemplo de Sesión Interactiva:

1. **Ver estadísticas:**

   ```
   > STATS
   +OK Statistics:
   +OK   Total connections:    1250
   +OK   Current connections:  45
   +OK   Bytes transferred:    10485760
   +OK End of statistics
   ```

2. **Listar usuarios:**

   ```
   > USERS
   +OK User list:
   +OK USER admin
   +OK USER invitado
   +OK End of user list
   ```

3. **Gestión de usuarios:**

```
 > ADDUSER nuevo_usuario pass_segura
+OK User added successfully

> DELUSER invitado
+OK User removed successfully
```

---

11. Documento de diseño del proyecto

Arquitectura Modular

El proyecto se estructura en módulos independientes con responsabilidades claras:

1. **Core (Framework):**
   - `selector.c` : Maneja el bucle principal de eventos. Registra descriptores de archivo (sockets) y despacha eventos de lectura/escritura a los handlers correspondientes.
   - `stm.c` : Motor de Máquinas de Estado Finito. Permite definir la lógica del protocolo como una secuencia de estados y transiciones.
2. **Protocolo SOCKS5 (`socks5nio.c`):** Implementa la lógica del proxy mediante una máquina de estados con las siguientes fases:
   - **HELLO:** Negociación de versión y métodos de autenticación.
   - **AUTH:** Lectura y validación de credenciales (RFC 1929).
   - **REQUEST:** Procesamiento del comando `CONNECT` .
     - *Sub-estado RESOLVING:* Si el destino es un dominio, se lanza un hilo para `getaddrinfo` y se pausa la conexión en el selector.
     - *Sub-estado CONNECTING:* Se inicia la conexión no bloqueante al *origin server*.
   - **COPY:** Estado de transferencia. Se leen datos del cliente y se escriben al origen, y viceversa. Se manejan cierres parciales de conexión (`shutdown`).
3. **Gestión (`mgmt.c`):** Implementa el servidor de administración. Utiliza su propia máquina de estados (`MGMT_AUTH` , `MGMT_CMD`) para procesar comandos de texto línea por línea.
4. **Servicios Compartidos:**
   - `buffer.c` : Gestión de buffers de memoria dinámicos para E/S.
   - `users.c` : Base de datos de usuarios thread-safe (usando mutex).
   - `metrics.c` : Colección de estadísticas usando variables atómicas (`_Atomic`) para garantizar consistencia sin bloqueos costosos.
   - `logger.c` : Sistema de registro de eventos y auditoría de accesos.

Diagrama de Estados Simplificado (SOCKS5)