Remote Process Manager

Internals de Sistemas Distribuidos y Programación POSIX

Equipo de Desarrollo

Ingeniería de Sistemas / Kernel Networking

6 de febrero de 2026

Contenido

Visión General de la Arquitectura

El sistema implementa una arquitectura **Cliente-Servidor** robusta basada en el estándar POSIX, diseñada para alto rendimiento y bajo consumo de recursos.

Componentes Principales

- **Lenguaje:** C (ISO/IEC 9899) para control total de memoria y sistema.
- **Transporte:** TCP/IP (Sockets de flujo) garantizando entrega de datos.
- **Modelo de Concurrencia:** Multihilo (`pthreads`) para atención simultánea.
- **Infraestructura:** Optimizado para despliegue en Linux (AWS EC2).

Ciclo de Vida de la Conexión (TCP)

El establecimiento de la comunicación sigue un flujo estricto para garantizar estabilidad:

1. **Inicialización:** Creación del Socket Stream (`AF_INET`, `SOCK_STREAM`).
2. **Binding:** Enlace al puerto 5002 en todas las interfaces (`INADDR_ANY`).
3. **Listening:** Cola de espera para conexiones entrantes.
4. **Handshake:** Aceptación de cliente (`accept`) y delegación.

Manejo de Concurrencia

Por cada cliente aceptado, el servidor dispara un **hilo dedicado** (`pthread_create`) en modo *detached*, permitiendo que el hilo principal siga escuchando nuevas peticiones sin bloqueo.

Intercambio de Mensajes

La comunicación se realiza mediante un protocolo de texto plano sobre TCP, diseñado para ser ligero y legible.

Estructura del Paquete:

- Buffer de Recepción: 64 KB
- Parsing: Tokenización por espacios
- Comandos: Verbo + Argumentos

Comandos Soportados:

- `LIST`: Consulta de tabla de procesos.
- `START <cmd>`: Ejecución remota.
- `STOP <pid>`: Terminación forzada.
- `EXIT`: Cierre de sesión.

Gestión de Buffers y Latencia

- **Buffering Extendido:** Se implementaron buffers de **65,536 bytes** para manejar salidas extensas de comandos del sistema (ej. listados de miles de procesos) sin fragmentación a nivel de aplicación.
- **Persistencia:** La conexión se mantiene viva (Keep-Alive implícito) permitiendo múltiples transacciones por sesión, reduciendo el overhead del handshake TCP.

Ejecución y Control de Procesos

El servidor interactúa directamente con el Kernel de Linux para gestionar tareas.

Comando START (Fork/Exec)

Utiliza la primitiva `fork()` para clonar el proceso servidor y `execvp()` para reemplazar la imagen de memoria con el nuevo comando.

- *Seguridad:* Redirección de `stdout/stderr` a `/dev/null` para evitar corrupción del socket.

Comando LIST (Pipes)

Abre un pipe de lectura (`popen`) ejecutando `ps -e -o pid,comm`, capturando la salida estándar del sistema en tiempo real.

Manejo de Señales y Limpieza

Para mantener la estabilidad del servidor durante ejecuciones prolongadas:

- **Prevención de Zombies:** Se implementó un manejador de señales (`sigaction`) para `SIGCHLD`.
- **Reap:** El servidor limpia automáticamente los recursos de los procesos hijos terminados usando `waitpid` con la bandera `WNOHANG`.

Compilación y Ejecución

Compilación (Makefile)

El proyecto incluye Makefiles optimizados para separar la lógica de cliente y servidor. `make -f Makefile.server` / `make -f Makefile.client`

Ejecución

- **Servidor:** `./server_bin` (Expone puerto 5002)
- **Cliente:** `./client_bin` (Requiere IP del servidor)

Repositorio: `https://github.com/riosisraelg/avanceProyecto`

Preguntas