🚀 Plan de Proyecto: Webserver C++98

Objetivo: Crear un servidor HTTP propio, no bloqueante, compatible con HTTP/1.1 y CGI.
Restricciones: C++98 estricto, sin librerías externas (Boost, etc.), gestión de memoria manual.

---

1. Arquitectura Técnica (Lo que hay que programar)

No dividiremos el trabajo en Front/Back. Dividiremos el servidor en 3 módulos lógicos para trabajar en paralelo:

- **Redes (Sockets & I/O):** Uso de socket, bind, listen. **Clave:** Multiplexación I/O no bloqueante con poll() (o epoll/kqueue). Un solo hilo para todo.
- **Protocolo (HTTP):** Lectura de RFCs. Parseo de texto crudo (GET /index.html...) y gestión de métodos (GET, POST, DELETE) y errores.
- **CGI (Ejecución dinámica):** Uso de fork, execve y pipe para ejecutar scripts (.php, .py) y devolver su salida.

---

2. Reparto de Roles (Equipo de 3)

Para aprender todos de todo, el diseño inicial (.hpp) lo hacemos juntos. Luego nos separamos:

👤 Integrante 1: El Core (Motor y Conexiones)

- **Misión:** Mantener el servidor vivo y aceptando miles de conexiones sin bloquearse.
- **Tareas:** Configurar el socket, gestionar el bucle infinito de poll(), detectar nuevos clientes y manejar buffers de lectura/escritura.
- **Reto:** Gestionar que la lectura no sea bloqueante (manejar datos parciales).

👤 Integrante 2: El Parser (Traductor HTTP)

- **Misión:** Dar sentido a los bytes que llegan.
- **Tareas:** Recibir texto crudo y convertirlo en una clase Request estructurada. Validar cabeceras y preparar la clase Response.
- **Reto:** Leer los RFCs, manejar *chunked transfer* y validar peticiones HTTP 1.1 correctamente.

👤 Integrante 3: La Lógica y Router (Gestor de Recursos)

- **Misión:** Decidir qué hacer con la petición (servir archivo, borrarlo o ejecutar script).
- **Tareas:** Leer archivo de configuración (puertos, rutas). Implementar la lógica de rutas y el sistema **CGI** (la parte más compleja de sistemas: pipes y forks).
- **Reto:** Conectar la entrada/salida de los scripts CGI sin crear *zombies* ni bloqueos.

---

3. Estrategia de Trabajo: "Desarrollo Aislado"

Para no bloquearnos unos a otros, trabajaremos con **Mocks** (simulaciones) hasta la integración final:

1. **Reunión CERO (CRÍTICA):** Definir juntos los archivos .hpp (Clases Request, Response, Config). Es el contrato que todos debemos respetar.
2. **Parser (Solo):** Crea un main de pruebas con strings de texto inventados para probar su parseo. No necesita red.
3. **Lógica (Solo):** Crea un main donde se inventa objetos Request ya limpios para probar si borra archivos o ejecuta el CGI.
4. **Core (Solo):** Hace que el servidor responda "HOLA MUNDO" a todo, para probar que aguanta 1000 conexiones simultáneas (stress test con telnet o siege).

---

4. Cronograma "Navideño" (Estimación: 6-8 Semanas)

🎅 Fase 1: Navidad y Año Nuevo (Diseño y Lectura)

- *Actividad:* Poca carga de código. Leer RFCs, entender poll y execve.
- *Meta:* **Tener definidos los .hpp antes de enero.** Si no tenemos las clases definidas, no podremos programar por separado.

👷 Fase 2: Enero - Semanas 3 a 5 (Implementación Dura)

- Vuelta a la rutina. Cada uno implementa su módulo aislado siguiendo la estrategia de Mocks.

🔥 Fase 3: Febrero - Semanas 6 y 7 (Integración y CGI)

- Unimos las partes. El Core pasa datos al Parser, el Parser al Router.
- *Punto crítico:* Debuguear el CGI y los Segfaults al juntar todo.

✨ Fase 4: Semana 8 (Pulido)

- Tests de fugas de memoria (Leaks/Valgrind) y cumplimiento de C++98.

💡 Resumen para sobrevivir

1. **No programar sin diseño:** Definamos las clases primero.
2. **No bloquearse esperando al otro:** Usad datos falsos para avanzar en vuestra parte.
3. **El Frontend:** Lo hacemos los tres en una tarde (HTML básico y feo), es lo de menos.
4. **Disfrutad las fiestas:** Aprovechad ahora para leer documentación. El código duro empieza el 7 de enero.