Análisis General del Servidor HTTP Mínimo para transferir el PDF

El programa establece un servidor TCP básico para la transferencia del documento holamundo.pdf.

Pasos 1 al 5: Conforman la rutina clásica de preparación de un socket servidor: se crea con socket(), se asocia al puerto con bind(), se pone en modo escucha con listen() y bloquea el hilo principal esperando la conexión de un cliente con accept().

Paso 6: Es vital para la interacción con un navegador. El navegador enviará una petición web (un GET / HTTP/1.1...); el servidor utiliza recv() para vaciar ese buffer de entrada y descarta el texto. Como el servidor tiene un único propósito (servir holamundo.pdf), no necesita parsear rutas ni parámetros.

Pasos 7 y 8: Abren el archivo PDF local, calculan su tamaño exacto usando fstat() y envían las cabeceras HTTP necesarias (HTTP/1.1 200 OK, Content-Type: application/pdf, Content-Length) a través del socket. Esto le indica al navegador el tipo MIME y el peso de la descarga para que muestre la barra de progreso correctamente.

Análisis Detallado del Paso 9: "Enviar el contenido del PDF (bucle robusto)"

Este bloque es fundamental en la programación de redes en C. Cuando trabajamos con la API de sockets, nunca debemos asumir que la función send() enviará el 100% de los bytes solicitados en una única llamada. Las redes, la congestión y los buffers internos del sistema operativo (OS) imponen límites en la cantidad de datos que pueden salir en un momento dado.

El Buffer (buf): Se declara un buffer estático de 8 KB (8192 bytes). Leer el archivo en fragmentos es una buena práctica para mantener el consumo de memoria RAM bajo (incluso si el PDF pesara cientos de megabytes) y para minimizar las llamadas al sistema (read()), que tienen un costo computacional.

```
char buf[8192];
```

Variable de lectura (r): Almacenará la cantidad de bytes que el sistema logró leer del disco en cada iteración.

```
ssize_t r;
```

Bucle de Lectura (Externo): La llamada intenta llenar el buffer con hasta 8192 bytes del archivo PDF.

Si r > 0, obtuvimos esa cantidad de bytes del disco y entramos al bloque para transmitirlos.

Si r == 0, llegamos al final del archivo (EOF) y el bucle termina, concluyendo la transferencia.

Si r < 0, ocurrió un error de lectura en el disco.

while ((r = read(pdf_fd, buf, sizeof(buf))) > 0)

Apuntador de desplazamiento (p): Apunta al inicio de la memoria que contiene el bloque de datos recién leído. A medida que los datos salgan por la red, este puntero avanzará.

char *p = buf;

Bytes restantes (left): Al iniciar la iteración, es igual a r (el total de bytes leídos que ahora debemos asegurar que lleguen al socket).

ssize_t left = r;

Bucle de Escritura (Interno): Aquí reside la verdadera robustez. Este while garantiza que el programa no saldrá a leer la siguiente parte del archivo hasta que los r bytes actuales hayan sido entregados completamente a la pila TCP del sistema operativo.

while (left > 0)

Llamada a la red: Se intenta enviar la cantidad de bytes indicada por left desde la posición de memoria p hacia el descriptor del cliente (fd_c). La función send() retorna la cantidad de bytes que *realmente* pudo encolar en el buffer de red (w). Si el buffer TCP está casi lleno, w será un número menor que left.

ssize_t w = send(fd_c, p, left, 0);

Manejo de Interrupciones (EINTR): Si send() devuelve un valor negativo, indica un error. Se inspecciona la variable global errno.

Si errno == EINTR (Error Interrupted), significa que la llamada bloqueante send() fue interrumpida por una señal asíncrona del sistema operativo antes de poder transferir ningún byte. La acción correcta frente a este evento no es abortar, sino reintentar. La instrucción continue obliga al bucle a volver a evaluar while (left > 0) e intentar el send() nuevamente.

Cualquier otro error (como que el alumno cierre la pestaña del navegador a la mitad de la descarga) provocará una salida de la función.

```
if (w < 0) {
   if (errno == EINTR)
      continue;
   perror("send(data)");
   // ... cierres de descriptores
   return EXIT_FAILURE;
}
```

Aritmética de Punteros:
Se resta la cantidad enviada (w) del total que falta por enviar (left).
Se avanza el puntero de memoria (p += w). De esta forma, si el bucle interno debe iterar nuevamente (porque send no envió todo en un solo envío), la siguiente ejecución comenzará a leer exactamente desde donde se quedó la anterior, transmitiendo solo la fracción restante del buffer de 8 KB.

```
left -= w;
p += w;
```