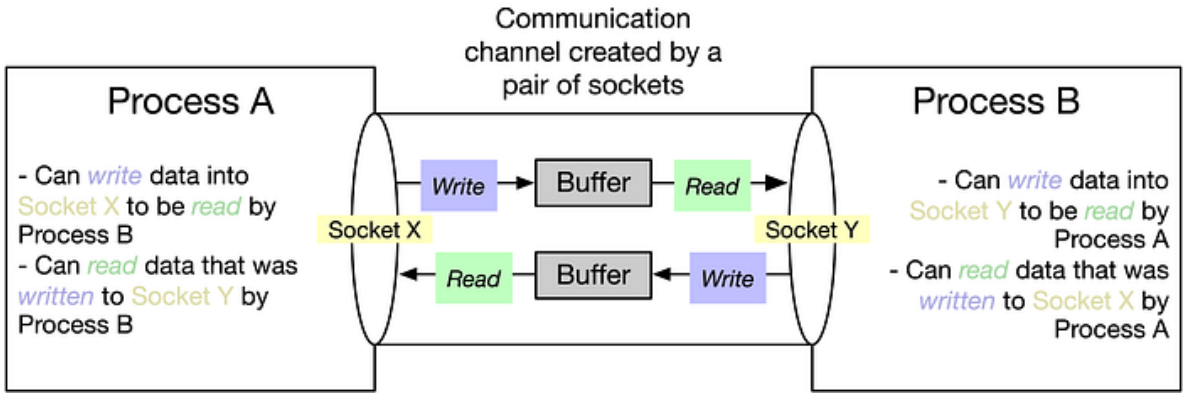
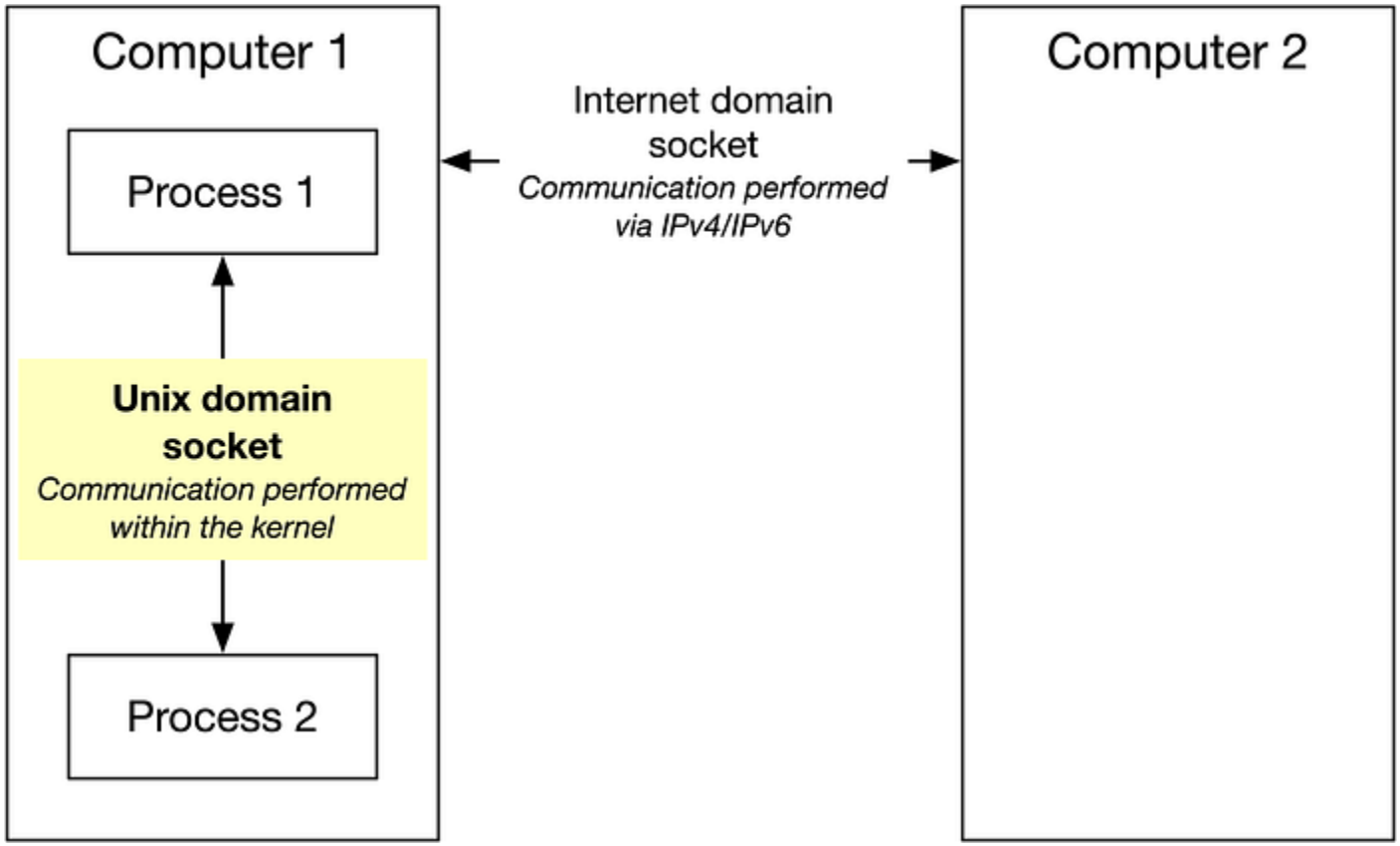
Los sockets AF_UNIX son un tipo de sockets de comunicación en Linux y otros sistemas operativos tipo Unix que permiten la comunicación interproceso (IPC, por sus siglas en inglés) entre procesos que se ejecutan en la misma máquina o host.



A diferencia de los sockets de red tradicionales, los sockets AF_UNIX utilizan una dirección de socket que se representa mediante un archivo del sistema de archivos. El archivo del socket es utilizado como un punto de conexión para la

comunicación entre procesos y puede ser creado y accedido utilizando funciones del sistema de archivos en lugar de llamadas de red.



Los sockets AF_UNIX son una alternativa a los sockets de red para la comunicación entre procesos locales, y son útiles para aplicaciones que requieren una comunicación de alta velocidad, bajo latencia y sin la sobrecarga de la red.

En resumen, los sockets AF_UNIX ofrecen las siguientes ventajas:

- Son una alternativa de comunicación interproceso a los sockets de red.
- Proporcionan una comunicación de alta velocidad y bajo latencia.
- Permiten la comunicación entre procesos que se ejecutan en la misma máquina o host.
- Utilizan archivos del sistema de archivos para representar la dirección del socket.

La estructura struct sockaddr_un

La estructura **sockaddr_un** es una estructura de datos en el lenguaje de programación C que se utiliza para representar direcciones de sockets UNIX. Esta estructura se utiliza junto con las llamadas del sistema para sockets UNIX para comunicar procesos en el mismo sistema o en sistemas diferentes a través de un sistema de archivos compartido.

La estructura **sockaddr_un** es definida de la siguiente manera:

```
struct sockaddr_un {
  sa_family_t   sun_family;      // Tipo de dirección, debe ser AF_UNIX
  char          sun_path[108];   // Nombre del archivo que representa el socket
};
```

sa_family_t es un campo que indica el tipo de dirección de la familia de direcciones que se está utilizando. En el caso de sockaddr_un, siempre es AF_UNIX.

sun_path es una cadena de caracteres que contiene el nombre del archivo que representa el socket.

La longitud máxima de **sun_path** es de 108 bytes. Si el nombre del archivo del socket es más largo que esto, se deben utilizar enlaces simbólicos para representar el nombre del archivo del socket.

Es importante tener en cuenta que la estructura **sockaddr_un** es específica para sockets UNIX y no se puede utilizar para otros tipos de sockets, como los sockets de red de Internet.

PROCESO SERVIDOR (“EL SERVIDOR”):

Este proceso vincula su socket a una ubicación conocida y acepta solicitudes de conexión entrantes de los clientes. Para cada solicitud de conexión que se recibe, se crea un nuevo socket que se usa para comunicarse con el socket en el otro extremo de la conexión, en este caso el socket creado por algún proceso cliente.

1. El servidor crea un nuevo socket utilizando la llamada al sistema socket(). Esto devuelve un descriptor de archivo que se puede usar para hacer referencia al socket en futuras llamadas al sistema.

2. El servidor utiliza la llamada al sistema bind() para vincular el socket a una dirección conocida, de modo que el cliente pueda conectarse a él.

3. El servidor llama a la llamada del sistema listen() para marcar el socket como pasivo, es decir, como un socket que aceptará solicitudes de conexión entrantes.

4. El servidor llama a la llamada del sistema accept() para aceptar una conexión entrante. **Esta llamada se bloquea** hasta que llega una solicitud de conexión. Tengan en cuenta que esta función generará un error si no se llama a listen() de antemano. Es importante destacar que esta llamada crea un nuevo socket que está conectado al socket par y devuelve un descriptor de archivo asociado con él. Por lo tanto, si desea comunicarse con el socket par, debe usar el descriptor de archivo devuelto por accept(), no el descriptor de archivo devuelto por la llamada a socket() en el paso n.º 1. El último socket permanece abierto y se utiliza para aceptar más solicitudes de conexión.

5. Después de esto, las llamadas al sistema read() y write() se pueden usar para comunicarse con el socket par (es decir, para comunicarse con el cliente).

6. Finalmente, cuando el servidor termine con el socket, debería llamar a close().

PROCESO CLIENTE ("EL CLIENTE"):

Este proceso conecta su socket a un socket servidor (o pasivo), después de lo cual es libre de comunicarse con el socket par (el socket cliente del lado del servidor). Tengan en cuenta que estos dos sockets son diferentes. El primero es el que se crea al llamar a socket() en el paso n.º 1 anterior, el último es el que se devuelve al llamar a accept() en el paso n.º 4 anterior.

1. El proceso cliente crea un nuevo socket utilizando la llamada al sistema socket(), que devuelve un descriptor de archivo que se puede usar para hacer referencia al socket en futuras llamadas al sistema.

   Uno socket creado con socket() se marca como activo y se puede usar en una llamada connect() para conectarse a un socket servidor (o pasivo).

2. El cliente llama a la llamada del sistema connect(), que se conecta a un socket servidor (o pasivo). Recuerda que el servidor vinculó su socket a una dirección conocida: esta es la dirección que debe usarse para connect().