**Documentación Practica 2**

Juan Manuel Alvarez  
Nicolás Campuzano

# Introducción

En el presente documento, se pasarán por cada una de las funciones que utilizan tanto el cliente como el servidor para su funcionamiento. Las 5 funciones básicas, que forman cada acción posible en el documento son, insertar una mascota, leer una mascota, leer su historia clínica, borrar una mascota y buscar una mascota por su nombre. Estas funciones serán detalladas con un diagrama de bloques y comunicaciones.

Al final, se hará hincapié en cómo el servidor puede trabajar con diferentes clientes al mismo tiempo.

# Insertar una mascota

**Cliente**

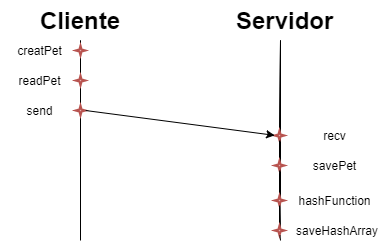
El usuario utiliza la función **createPet** en la cual se hace se crea una struct dogType donde estarán todas propiedades que el usuario escribirá una por una en el método **readPet**. Este último método, entrega un dogType con sus propiedades el cual se envía al servidor, a través de la función **send**, propia de los sockets.

**Servidor**

En el servidor, la acción que inicia el insertar una mascota proviene de un request tipo **INSERCION**, la cual llama a **createPet** con un puntero dogType como parámetro. Allí, tenemos varios pasos, enunciados a continuación:

1. Bloquear a través de mutex, por si cualquier otro request quiere acceder a los archivos afectados.
2. Guardar la mascota con la función **savePet**, la cual escribe la mascota en el archivo **dataDogs.dat**
3. Aumentar dogAmount la cual se guarda en el **petAmount.dat**
4. Asignarle el hash correspondiente con la función **hashFunction** y **changeHashHead** que recibe el nombre del perro como parámetro.
5. Guardar el nuevo arreglo a través de la función **saveHashArray** en el archivo **hashDog.dat**
6. Desbloquear el mutex para que los demás hilos continúen su ejecución.

**Diagrama de comunicaciones**



# Leer una mascota

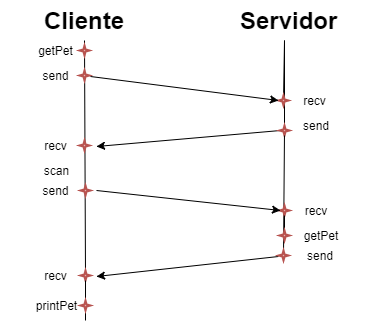
**Cliente**

Para este caso, la función **getPet** es la que se encarga de mostrar en pantalla una mascota. En primer lugar, utiliza la función **send** para preguntarle al servidor cuántos perros existen, y con la función **recv** recibe y lo guarda en la variable **dogAmount**. De esta manera el usuario puede seleccionar un id de perro válido, el cual es enviado de nuevo con la función **send** y un request con acción **LECTURA**. Se recibe el perro con la función **recv** y se guarda en la variable **pet** que será pasada a **printPet**, la cual imprime en pantalla un **struct dogType.**

**Servidor**

En el servidor, la acción que inicia el insertar una mascota proviene de un request tipo **LECTURA**, la cual llama a **getPet** con un int index como parámetro. En este caso la función **getPet** devuelve la información de un struct dogType que será enviado al cliente. Esta información es leída del archivo **dataDogs.dat**.

**Diagrama de comunicaciones**



# Imprimir historia clínica

**Cliente**

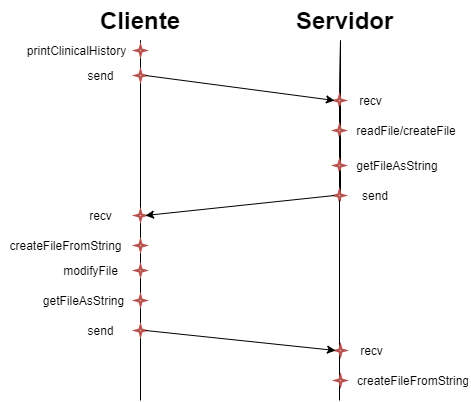
El usuario utiliza la función **printClinicalHistory,** una vez encuentra el perro que le interesa mediante la función **getPet,** pidiéndole al servidor que le envié la historia clínica como esta hasta el momento. Una vez la recibe, crea un archivo temporal externo al programa para editarlo a su gusto. Una vez finalizada la edición, le envía la cadena de caracteres final al Servidor para que lo almacene.

**Servidor**

En el servidor, la acción que inicia el imprimir la historia clínica de una mascota proviene de un request tipo **HISTORIA**, la cual llama a **printClinicalHistory** con el índice del perro. Allí, tenemos varios pasos, enunciados a continuación:

1. Bloquear a través de mutex, por si cualquier otro request quiere acceder a los archivos afectados.
2. Si la historia clínica del perro solicitado no existe, crea una historia clínica modelo con sus datos básicos.
3. Transforma la historia clínica del perro a un String, leyendo el archivo.
4. Envia el String al cliente para ser editado y espera a que le regrese la versión modificada.
5. Cuando recibe la String del cliente, reemplaza el archivo de historia clínica existente por el modificado con la String del usuario
6. Desbloquear el mutex para que los demás hilos continúen su ejecución.

**Diagrama de comunicaciones**



# Borrar mascota

**Cliente**

El usuario utiliza la función **deletePet**, primero preguntará cuántos perros hay en elservidor, para que así el usuario ingrese un número válido a borrar. Luego a través de la comunicación con el socket, envía un index para que éste borre el perro.

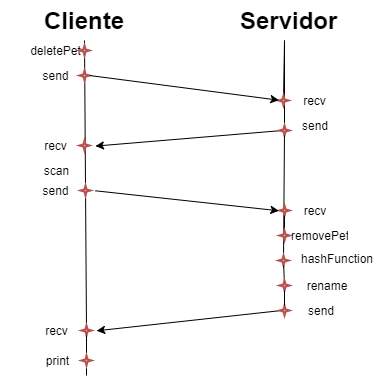
**Servidor**

El servidor comienza el procesamiento de esta función luego que reciba un request con acción **BORRADO**. Llama a la función **removePet** enviándole un index que fue recibido en la función **recv**. La función removePet será descrita a continuación

1. Bloquear a través de mutex, por si cualquier otro request quiere acceder a los archivos afectados.
2. Remover la cabeza de hash en el arreglo de hash y reemplazarla por la siguiente.
3. Reformar el hash dependiendo si es un perro que está en la mitad, de la lista de hash, al inicio o al final.
4. Borrar la historia clínica en caso de que exista.
5. Copiar del archivo anterior a un nuevo archivo temporal, sin el dato a borrar. Posterior a esto, renombrar el archivo temporal y borrar el anterior, de esta manera se reduce el tamaño del archivo.
6. Desbloquear el mutex para que los demás hilos continúen su ejecución.

Luego se envía un mensaje de que el perro fue borrado correctamente o hubo algún fallo.

**Diagrama de comunicaciones**

****

# Buscar una mascota

**Cliente**

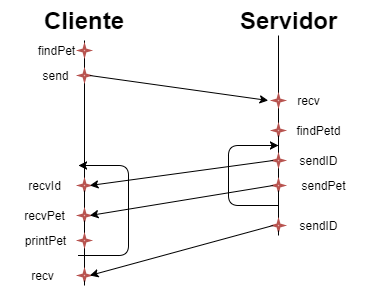
El usuario utiliza la función **findPet**, la cual le pide al usuario que ingrese un nombre de 32 caracteres, el cual se le envía al servidor. El usuario primero recibe el ID del perro, si es -1 quiere decir que no se encontró ninguna coincidencia y si es -2 entonces que se terminó la búsqueda. Por cada dato recibido lo imprime en pantalla.

**Servidor**

El servidor comienza el procesamiento de esta función luego que reciba un request con acción **BUESQUEDA**. Llama a la función **findPet** enviándole la información de cliente, la cadena a buscar y una variable auxiliar. Esta función tiene los siguientes pasos:

1. Bloquear a través de mutex, por si cualquier otro request quiere acceder a los archivos afectados.
2. Obtener el hash de la cadena a buscar.
3. Iterar por cada perro que tenga el mismo hash encontrado.
4. Volver tanto el nombre del perro como de la cadena, a minúsculas para que sea case insensitive.
5. Por cada perro encontrado enviar el ID del perro primero, y luego el perro para imprimir. Si no hay perros se enviará un -1 indiciando que no hubo perros encontrados.
6. Luego enviarle al cliente un entero -2 que indicia que se acabó la búsqueda.

**Diagrama de comunicaciones**



# Funcionamiento múltiples clientes

Para que múltiples clientes funcionen el servidor cada vez que la función **accept** se ejecute, creará un nuevo hilo, el cual tendrá la información del cliente. La máxima capacidad de clientes en este momento son 8 para 8 hilos del procesador. Al final se recorren por los 8 hilos haciendo un **join** de ellos, con lo cual se busca esperar a que terminen antes de que el programa cierre.

Por otro lado, cada función que utilice archivos o recursos críticos del programa tiene bloqueado a través de mutex la ejecución de los otros hilos.

# Generación de Logs

Para las 5 funciones esenciales del programa, antes de ejecutar la función se pasa por la función **makeLog** la cual guarda en el archivo **serverLogs.log** un log donde está el tipo de request, la IP del cliente, la hora de petición y el ID o palabra a buscar en caso tal.