Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

NX-Engeenering

NuttX remote client: Implementación de un cliente para protocolos de distribución remota para el NuttX RTOS.

Informe de Proyecto de Graduación para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura/Bachillerato

Jose Pablo Rojas Vargas

Cartago,
Junio, 2012

TEC Tecnológico de Costa Rica

INSTITUTO TECNOLOGICO DE COSTA RICA

ESCUELA DE INGENIERIA ELECTRONICA

PROYECTO DE GRADUACIÓN

TRIBUNAL EVALUADOR

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica

Miembros del Tribunal

Ing. Johan Carvajal Godínez, M.Eng
Profesor Asesor

Dr. Carlos Meza Benavidez
Profesor lector

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica

Cartago, Junio 2012

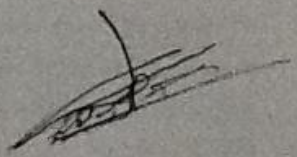
Declaración de autenticidad

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía, he procedido a indicar las fuentes mediante las respectivas citas bibliográficas.

En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Cartago, Junio 2012



Jose Pablo Rojas Vargas

Cédula: *1-11359-0466*

ii

Resumen

En este documento se presenta el estudio realizado para la implementación de una nueva función que pueda acceder a ficheros contenidos en una red de área local.

El proyecto se enfocó en un protocolo de distribución de datos remoto el cual toma un sistema de archivos especial que, a la hora de ser instalado, requiere de información de red para operar. Una vez que esto sucede y se conecta a un servidor, puede ser usado desde el mismo por la tarjeta embebida con el sistema operativo como cualquier otro archivo local.

El protocolo escogido fue el desarrollado por Sun Microsystems, cuyo nombre es (NFS) “Network File System”, que permite a un usuario en un equipo cliente acceder a archivos en una red local de una manera similar a como se accede al almacenamiento local.

Durante el desarrollo del proyecto se tomó como base el código fuente de los sistemas operativos FreeBSD y OpenBSD para la elaboración completa de la nueva función del NuttX RTOS, el cual cuenta con una licencia de software libre permisiva para utilizar estos códigos. Estos dos sistemas operativos son distribuciones de código abierto son pertenecientes a BSD o “Berkeley Software Distribution”, sistema operativo derivado del sistema UNIX, el cual nace de los aportes realizados por la Universidad de California en Berkeley.

Al final del desarrollo del proyecto, se logró Implementar un cliente NFS para el NuttX RTOS, el cual permite acceder archivos remotos como si estos fueran locales, realizando al menos las operaciones básicas del protocolo NFS, así como la obtención de un diseño enfocado al uso óptimo de la memoria, ya que el sistema de archivos de red utiliza siempre un valor constante de memoria igual a 2192 kBytes.

Palabras claves: NFS, RPC, VFS, NuttX RTOS, NSH.

Abstract

This paper presents the study of the implementation for a new feature that can access files in a local area network.

The project is focused on a protocol for remote data distribution which takes a special file system, and when it is installed, network information is required to operate. Once this happens and it’s connected to a server, it can be used from the embedded board with the operating system as any local file.

The chosen protocol was developed by Sun Microsystems, whose name is (NFS) "Network File System", which allows a client user to access files on a local network like a similar manner of accessing the local storage.

The development of the project was based on the source code of the FreeBSD operating system and OpenBSD for the new role of NFS, because it has a permissive free software license to use these codes in NuttX RTOS. These two operating systems are open source distributions owned by BSD or "Berkeley Software Distribution", operating system derived from the UNIX system, which arises from the contributions made by the University of California at Berkeley.

At the end of project, it was achieved the Implementation of a NFS client for NuttX RTOS, which lets you access remote files as if they were local, making at least the basic operations of the NFS protocol and obtaining a design focused on the use optimal memory because the network file system always uses a constant value equal to 2192 kBytes memory.

Keywords: NFS, RPC, VFS, NuttX RTOS, NSH.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1 Introducción

1.1 Problema existente e importancia de su solución

El presente proyecto de graduación se llevó a cabo con la empresa NX-Engineering, la cual dio inicio a sus operaciones desde el 2008. Desde entonces, el fundador Gregory Nutt ha estado trabajando alrededor del sistema operativo de tiempo real NuttX RTOS, sistema que él mismo creó, y con el que ha tenido mucho éxito debido a características únicas que posee.

El desarrollador del NuttX RTOS se percató de una oportunidad de mejora en el mismo, ya que el sistema no contaba con funciones que permitieran el intercambio de ficheros a través de la red, optimizando el uso de la memoria RAM de la tarjeta embebida.

De ahí, que surge la idea de desarrollar esta funcionalidad en el NuttX RTOS y probarla en un sistema embebido con poca capacidad de memoria RAM para validar la funcionalidad.

El tamaño de la memoria que requiere el NuttX RTOS, es una de las principales características que lo hace ideal para entornos con micro-controladores y es totalmente escalable en sistemas pequeños (8 bits), así como en sistemas más potentes (32 bits). Aunado a esto, el NuttX RTOS cumple con normas y estándares generales de sistema operativo de tiempo real (RTOS), y posee código abierto, por lo que mejoras continuas son realizadas, posibilitando al sistema operativo ser lo suficientemente versátil en varios tipos de plataformas de hardware.

Con lo anterior, se hace clave que con el crecimiento de las capacidades del sistema operativo, se mantengan las características de requerimiento de memoria para las plataformas a las que se pretende portar el NuttX. Para ello, se hace necesario se cuente con capacidad de acceso remoto de ficheros, mediante

la implementación de un protocolo de distribución de datos remoto con la capacidad de acceder y mapear archivos que se hospedan en un servidor, y que estos puedan ser modificados y utilizados como si estuvieran hospedados localmente en la tarjeta embebida.

1.2 Solución seleccionada

El proyecto realizado se presenta como un aporte significativo en el área de los sistemas empotrados, específicamente en el área de comunicación.

Para la realización de esta nueva función, se decidió utilizar el protocolo NFS, que permite la distribución de datos remoto.

El proyecto tomó inicialmente como base el protocolo CIFS, esto porque la mayoría de los usuarios desarrollan en el ambiente Windows, no obstante se encontró una limitante en el acceso a la información; esta tecnología no data de hace mucho tiempo, por lo que al no contar con todos los insumos necesarios se optó por implementar el sistema por medio del protocolo NFS, que tiene su utilidad en servidores UNIX.

Se cree necesario explicar, a *grosso modo,* lo que significan estos conceptos anteriormente mencionados. El Network File System (NFS) es un protocolo de sistema de archivos de red desarrollado originalmente por Sun Microsystems en 1984, que permite a un usuario en un equipo cliente acceder a archivos en una red local de una manera similar a como se accede al almacenamiento local. El sistema de archivos de red es un estándar abierto definido en el RFC, permitiendo que cualquiera pueda implementar el protocolo. [5]

Por el contrario, el Common Internet File System (CIFS) es el protocolo basado en Windows usado en contraparte para el intercambio de archivos. CIFS es en realidad la versión pública de SMB (Server Message Block Protocol), creado por Microsoft. Este protocolo permite el intercambio de archivos de

múltiples dispositivos, como impresoras, archivos, e incluso los puertos serie, entre los diversos usuarios y administradores. [1]

Para su implementación, el sistema tenía como especificaciones mínimas, el uso de un microprocesador que ejecute el NuttX RTOS, así como una interfaz de red por medio de Ethernet para conectarse a la red local, por lo que se utilizaron dos tarjetas de desarrollo que sean compatibles con el NuttX RTOS, como lo es el caso de la TI/Luminary Stellaris LM3S6965 Evaluation Board [4] y la EAGLE 100 Single Board [6]. Estas tarjetas poseen un microprocesador basado en el ARM Cortex-M3 [21] que trabaja a 50 MHz, una memoria SRAM de 64 kB y además poseen una interfaz de 10/100Ethernet, lo que permitirá al módulo conectarse a internet mediante cualquier red con esta tecnología. . La figura 1.1 y 1.2 muestran el hardware seleccionado para el desarrollo del proyecto.

Figura 1.1 Tarjeta de evaluación Stellaris LM3S6965 [4].

Figura 1.2 Tarjeta Eagle 100 de Micromint [6].

Capítulo 2 Meta y objetivos

2.1 Meta

Realizar una nueva función de cliente de datos remotos para el NuttX RTOS, que permita reducir el tamaño de la memoria usada en el sistema empotrado en al menos un 25% de su tamaño actual.

2.2 Objetivo general

Desarrollar un cliente de acceso de datos en el NuttX RTOS, mediante la implementación de un protocolo de distribución de archivos, de manera que desde un sistema embebido se puedan acceder archivos remotos como si estos fueran locales.

2.3 Objetivos específicos

- Seleccionar el protocolo de acceso remoto de archivos, que mejor se adapte a las características de hardware de los sistemas embebidos que ejecutan el sistema operativo de tiempo real NuttX.
- Desarrollar los algoritmos necesarios para la implementación eficaz del protocolo de distribución de archivos remotos sobre el NuttX RTOS sobre una plataforma embebida de referencia, el cual permita crear, modificar y eliminar un archivo que se hospeda remotamente en servidor.
- Evaluar la compatibilidad de la nueva función de cliente en dos plataformas que ejecuten el NuttX RTOS accediendo a archivos remotos como si estos fueran locales mediante el uso de un protocolo de distribución remota.

- Optimizar el espacio en memoria del cliente de acceso remoto de datos, de forma tal, que no supere 5 KB de espacio en memoria RAM del sistema embebido.

.

Capítulo 3 Marco teórico

3.1 Descripción del sistema

Inicialmente, Nuttx RTOS posee sistemas de archivos, como FAT y ROM, que no ocupan una conexión de red ya que todo el manejo interno se realiza de manera local, permitiendo que el sistema operativo corra perfectamente en los dispositivos de almacenamiento contenidos en la tarjeta embebida.

La función que se desarrolló en este proyecto, de cliente NFS, permitió introducir un nuevo sistema de archivos que utiliza una conexión de red para su funcionamiento. Como se muestra en la figura 3.1, se elaboró de forma interna en el NuttX RTOS.

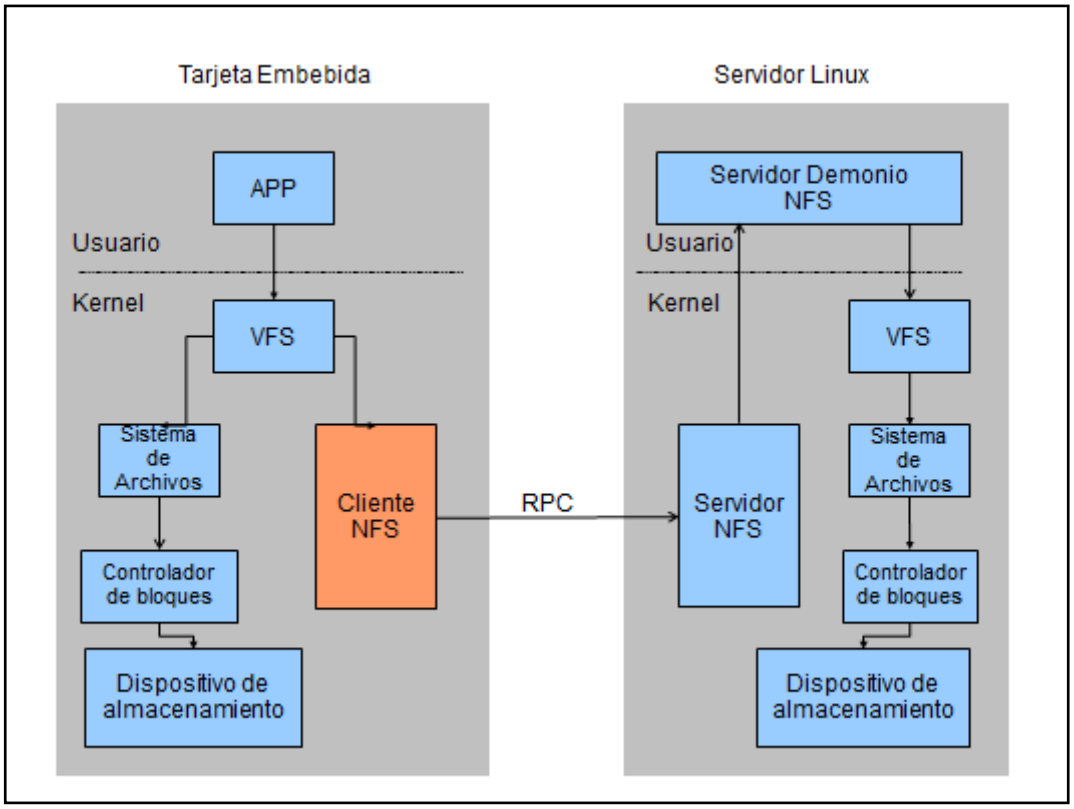


Figura 3.1 Diagrama de bloques del sistema

La función utiliza una aplicación que es interactuada por el usuario cuyo nombre es NSH (ver sección 3.1.1), la cual, mediante la introducción de una serie de comandos, permite el montaje y desmontaje del sistema de archivos NFS (ver

sección 3.1.2) en la tarjeta embebida, así como las diferentes funciones sobre los archivos presentes en un servidor remoto. Esto se realiza mediante el envío de distintos mensajes que utilizan el protocolo RPC (ver sección 3.1.3) hacia el demonio del servidor NFS, que se encuentra esperando mensajes. Es importante recalcar, que todo cambio dentro y fuera de los archivos es llevado a cabo desde el servidor, enviando así información al cliente sobre su estado.

El agregado de este sistema de archivos se hace más fácil gracias a que en NuttX RTOS se encuentra implementado el VFS (ver sección 3.1.4), lo que permite que una aplicación pueda acceder a distintos sistemas de archivos concretos sin notar las diferencias entre ellos, es decir, que utiliza un manejo común exponiendo la versatilidad del sistema.

3.1.1 NuttShell (NSH)

El “NSH” o NuttShell se refiere a un pequeño y escalable, intérprete de comandos para el NuttX RTOS que presenta un conjunto de características. [11]

Si nos introducimos en el sistema operativo, en el sub-directorio *apps/nshlib* se logra ubicar la biblioteca del NuttShell (NSH). Esta biblioteca puede ser fácilmente enlazada para producir así la aplicación de NSH. Este intérprete es una aplicación de “shell” simple para el NuttX RTOS.

Cuando tenemos la aplicación ya generada, se puede utilizar los ajustes en el archivo de configuración, para que de esta manera NSH se pueda configurar ya sea para utilizar la entrada estándar serial o una conexión telnet como una consola o simplemente funcionar de las dos maneras. Cuando el NSH se inicia, se puede observar la siguiente bienvenida:

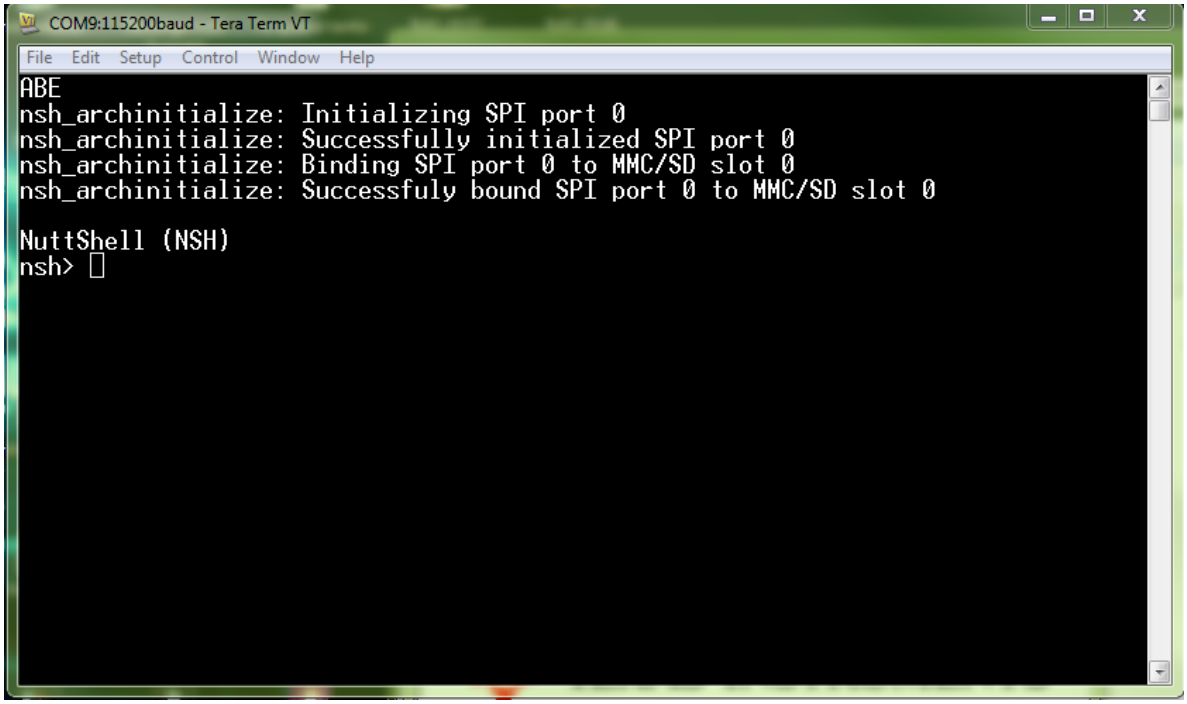


Figura 3.2 Consola del NuttShell (NSH).

nsh> indica que usted puede introducir un comando de la consola.

Este soporta 35 comandos, entre algunos de estos soportados por NSH se tienen los siguientes:

- test : Evalúa variables
- cp : Copia archivos
- free : Muestra el estado del manejo de memoria
- ls : Lista el contenido del directorio
- mkdir : Crea un directorio
- rm : Remueve un archivo
- rmdir :Remueve un directorio
- mount : Monta un sistema de archivos
- umount : Desmonta un sistema de archivos
- cat : Concatena archivos.

3.1.2 Network File System (NFS)

NFS es un protocolo basado en RPC, diseñado para soportar llamadas a procedimientos remotos, con una relación de cliente-servidor entre la máquina que tiene el sistema de archivos para su distribución y la máquina que quiera tener acceso a ese sistema de ficheros. Las especificaciones del protocolo se pueden encontrar en el RFC número 1813 [20]
Los servidores NFS corren en ellos servicios que permiten tramitar las solicitudes del sistema y alguna traducción de ruta.

Un cliente NFS puede montar la totalidad o parte de un sistema de archivos remoto. Este puede acceder a los archivos de este sistema de archivos como si estuvieran presentes en un disco duro local.

Todas las operaciones de NFS se implementan como procedimientos de RPC. Un resumen de los procedimientos de NFS se muestra a continuación:

- Procedimiento 0: NULL - No hacer nada.
- Procedimiento 1: GETATTR - Obtener atributos de un archivo.
- Procedimiento 2: SETATTR - Establecer los atributos de un archivo.
- Procedimiento 3: LOOKUP - Búsqueda de nombre de un archivo.
- Procedimiento 4: ACCESS - Chequeo de permiso de acceso.
- Procedimiento 5: READLINK - Leer de enlace simbólico.
- Procedimiento 6: READ - Leer un archivo.
- Procedimiento 7: WRITE - Escribir un archivo.
- Procedimiento 8: CREATE - Crear un archivo.
- Procedimiento 9: MKDIR - Crear un directorio.
- Procedimiento 10: SYMLINK - Crea un enlace simbólico.
- Procedimiento 11: MKNOD - Crear un dispositivo especial.
- Procedimiento 12: RMOVE - Eliminar un archivo.
- Procedimiento 13: RMDIR - Quitar un directorio.
- Procedimiento 14: RENAME - Cambiar el nombre de un archivo o directorio.
- Procedimiento 15: LINK - Crear un enlace a un objeto.

- Procedimiento 16: READDIR- Leer un directorio.
- Procedimiento 17: READDIRPLUS - Lectura ampliada del directorio.
- Procedimiento 18: FSSTAT - Obtener información dinámica del sistema de archivos.
- Procedimiento 19: FSINFO - Obtener información estática de sistema de archivos.

3.1.3 Remote Procedure Call (RPC)

El protocolo RPC de Sun se describe específicamente en el RFC número 1831 [22], este es un protocolo para solicitar un servicio de un programa ubicado en un ordenador remoto a través de la red sin tener que entender la virtud de las tecnologías de capa de red. Sun RPC a veces se le llama ONC (Open Network Computing) de RPC. Los desarrolladores que implementan el estándar tienen la opción de usar llamadas a procedimiento remoto sobre cualquiera de los protocolos de transporte UDP/IP o TCP/IP.

Protocolo trabaja de la siguiente manera: Existe un programa solicitante que es el cliente y un programa de proveedor de servicios el cual es el servidor. Inicialmente, el proceso emisor envía un mensaje de llamada que incluye los parámetros de procedimiento para el proceso del servidor. Así, el proceso que llama, espera un mensaje de respuesta. A continuación, un proceso en el lado del servidor, que es inactivo hasta la llegada del mensaje de llamada, extrae los parámetros del procedimiento, calcula los resultados, y envía un mensaje de respuesta. El servidor espera un próximo envió del mensaje. Finalmente, un proceso en el que llama, recibe el mensaje de respuesta, extrae los resultados del procedimiento, y el que llama reanuda la ejecución.

Es importante mencionar, que existen protocolos de unión que van de la mano con el protocolo RPC, como lo son MOUNTD y PMAP.

PMAP es un protocolo que se encarga de mapear programas y servicio que utilizan RPC, asignándoles así un número de puerto específico. Mientras que MOUNTD es un servicio que está esperando alguna petición de un cliente para efectuar un montaje de un sistema de archivos.

3.1.4 Virtual File System (VFS)

Virtual File System (VFS) o Virtual Filesystem Swith se trata de una capa abstracta que se utiliza sobre un sistema de archivos concreto. Esta abstracción tiene el fin de que las aplicaciones tengan el acceso a diversos tipos de sistemas de archivos concretos de una manera uniforme. De esta manera es posible y fácil tener varios sistemas de archivos en un kernel si estos cumplen con la compatibilidad de esta capa de abstracción, ya que el VFS especifica un tipo de contrato entre los sistemas de archivos y el kernel.
Un VFS describe un sistema de archivos en términos de un “Inode”. Un VFS “Inode” describe archivos y directorios, es decir, el contenido de un sistema de archivos.

3.2 Antecedentes Bibliográficos

NFS fue desarrollado dentro de Sun Microsystems a principios del año 1980. Desde ese momento, NFS ha sido objeto de tres revisiones importantes:

NFS v1:

NFS versión 1 era un prototipo del sistema de archivos de red de la empresa Sun. Esta versión nunca fue lanzada al público.

NFS v2:

NFS versión 2 se distribuyó por primera vez con el sistema operativo de Sun “SunOS 2” en 1985. La versión 2 fue autorizada ampliamente a numerosos proveedores de estaciones de trabajo UNIX. Posteriormente, una distribución del

tipo gratuita y compatible, se desarrolló a finales de los 80’s en la Universidad de California en Berkeley.

Durante sus 10 años de vida, se realizaron muchos cambios sutiles indocumentados, que fueron introducidos en las especificaciones de NFS v2. Algunos proveedores de NFS v2 permitieron la lectura o escritura de más de 4K bytes en un momento, mientras que otros aumentaron el número de grupos previstos en el marco de la autenticación de RPC desde 8 a 16. A pesar de estos cambios menores creó incompatibilidades ocasionales entre diferentes implementaciones de NFS, pero de igual manera NFSv2 proporciona un notable grado de compatibilidad entre los sistemas hechos por diferentes fabricantes.

NFS v3:

La versión 3 de NFS especificación fue desarrollada en una serie de reuniones en Boston en julio de 1992. El Código de Trabajo para NFS v3 fue presentado por algunos vendedores en el año 1995. Esta incorpora muchas mejoras de rendimiento sobre la versión 2, pero no cambia significativamente la forma en que las obras de NFS o el modelo de seguridad utilizado por el sistema de archivos de red.

3.3 Descripción de los principales principios relacionados con la solución del problema

En este proyecto el uso de un modelo de referencia es de suma importancia ya que este brinda una referencia común para mantener consistencia en todos los tipos de protocolos y servicios de red. Tal como se ha visto, el protocolo NFS provee un acceso remoto de forma transparente para compartir archivos a través de la red. Este logra establecer esta forma de transparencia gracias al uso del protocolo RPC, que a su vez luego se hace una conversión de los datos de manera que los datos estén listos para ser enviados.

3.3.1 Capa OSI

El modelo de referencia que se utilizó para el proyecto es el modelo de Interconexión de Sistemas Abiertos (OSI) establecido por la Organización Internacional de Estándares (ISO). Este modelo es un set de funciones comunes en el entorno de red para la transferencia de datos [18], el cual está estructurado en siete capas, tal como se muestra a continuación:

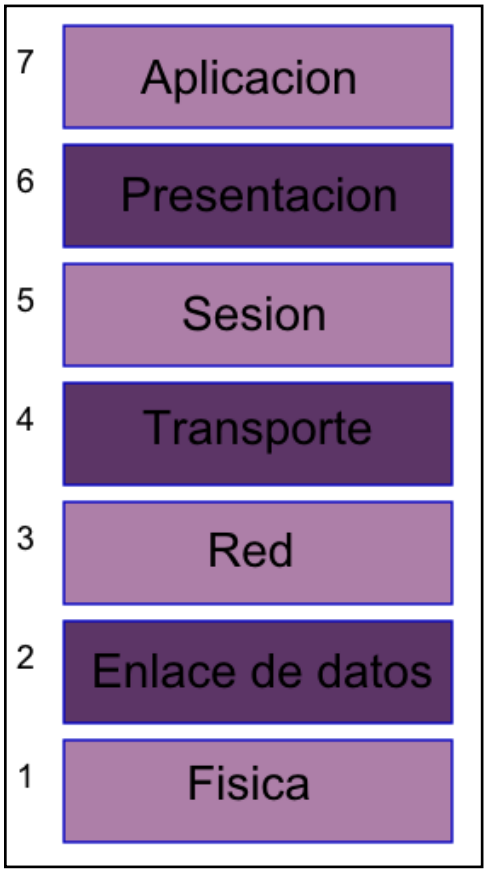


Figura 3.3 Modelo OSI

Capa de Aplicación: Es la capa que proporciona la interfaz entre las aplicaciones que se utilizan para comunicar y la red subyacente en la cual se transmiten los mensajes.

Capa de Presentación: Es la capa que se encarga de la representación de la información, de manera que aunque distintos equipos puedan tener diferentes representaciones internas de caracteres los datos lleguen de manera reconocible

Capa de Sesión: Las funciones en esta capa crean y mantienen diálogos entre las aplicaciones de origen y destino.

Capa de Transporte: Esta capa mantiene el control de flujo de datos, y provee de verificación de errores y recuperación de datos entre dispositivos. Control de flujo significa que la capa de transporte vigila si los datos vienen de más de una aplicación e integra cada uno de los datos de aplicación en un solo flujo dentro de la red física.

Los dos protocolos más comunes de la capa de Transporte son el Protocolo de control de transmisión (TCP) y el Protocolos de datagramas de usuario (UDP). El primero orientado a conexión y el otro sin conexión. Ambos protocolos gestionan la comunicación de múltiples aplicaciones.

Capa de Red: Esta capa determina la forma en que serán mandados los datos al dispositivo receptor. Aquí se manejan los protocolos de enrutamiento y el manejo de direcciones IP.

Capa de Enlace de Datos: La capa de enlace de datos proporciona un medio para intercambiar datos a través de medios locales comunes. Es decir, la capa de enlace de datos se ocupa del direccionamiento físico, de la topología de la red, del acceso a la red, de la notificación de errores, de la distribución ordenada de tramas y del control del flujo.

Capa Física: La función de la capa física de OSI es la de codificar los valores binarios que representan las tramas de la capa de Enlace de datos en señales, además se encarga de transmitir y recibir estas señales a través de los medios físicos como alambres de cobre, fibra óptica o medios inalámbricos que conectan los dispositivos de la red.

3.3.2 Forma de los datos

Dentro del modelo OSI, los datos reciben una serie de nombres y formatos específicos en función de la capa en la que se encuentren, esto porque dentro de cada capa se posee una serie de encabezados e información final distinta. Los formatos de información se muestran a continuación:

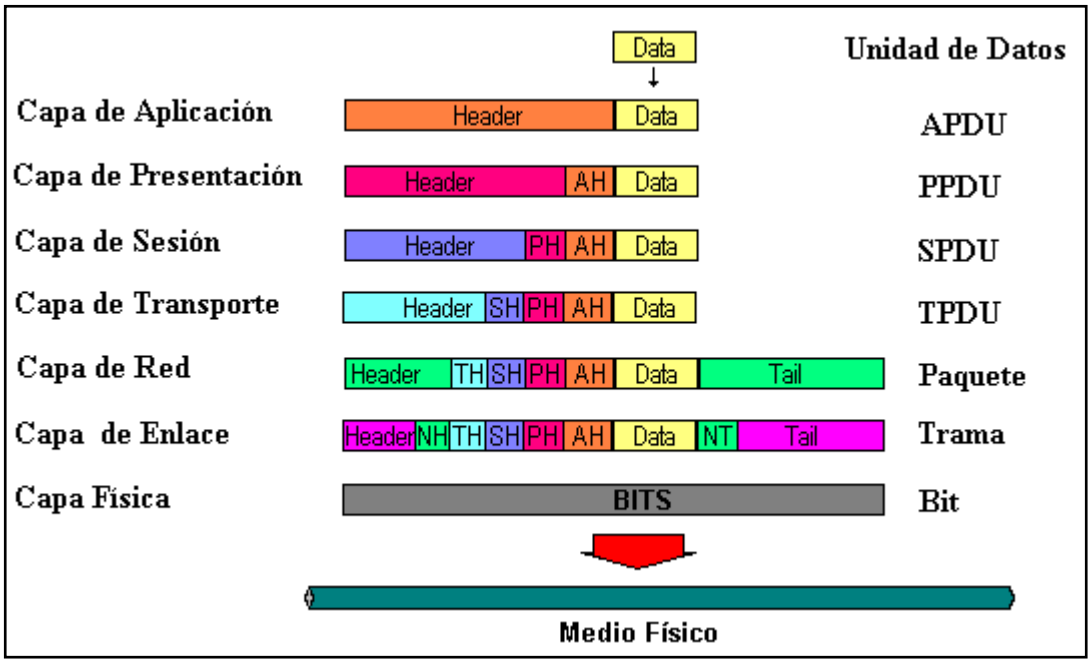


Figura 3.4 Unidad de datos en cada capa del modelo OSI. (Ver apéndice A.1).

Capítulo 4 Metodología

El proyecto "*NuttX remote client: Implementación de un cliente en una tarjeta de desarrollo utilizando un protocolo de distribución remota para el NuttX RTOS*" surge debido a que el RTOS no cuenta con funciones que permitan el uso, la modificación y el manejo de documentos a través de la red, las cuales optimizan el uso de la memoria RAM de la tarjeta.

La ejecución del proyecto se dividió en varias etapas. La primera etapa consistió en el estudio del documento RFC 1813 [20], que posee las especificaciones, conceptos, la estructura y el funcionamiento del protocolo NFSv3.

Seguidamente, aprovechando que el NuttX RTOS cuenta con una licencia de software libre total permisiva de BSD, se optó por buscar las implementaciones del protocolo NFS referentes a las distribuciones de BSD, como el FreeBSD y el OpenBSD. De esta manera, con base en el código fuente de estas dos distribuciones se implementó en proyecto en lenguaje C.

Finalmente se realizó una depuración a nivel general, para determinar y solucionar los diferentes errores en la implementación del diseño, así como analizar el código, de forma que, este optimizara el uso de la memoria.

Capítulo 5 Descripción detallada de la solución

A continuación se describe la implementación de la solución propuesta, donde se detalla la estructura del diseño, así como la de su funcionamiento. Además, se realiza una descripción puntualizada de las diferentes rutinas de software desarrolladas que permiten el funcionamiento de la nueva función del protocolo NFS, así como la descripción del hardware que se utilizó para la realización del proyecto.

5.1 Descripción del hardware

En esta sección se detalla el hardware utilizado en la implementación del proyecto. Este consta de dos partes o módulos tal como se muestra a continuación en la figura 5.1

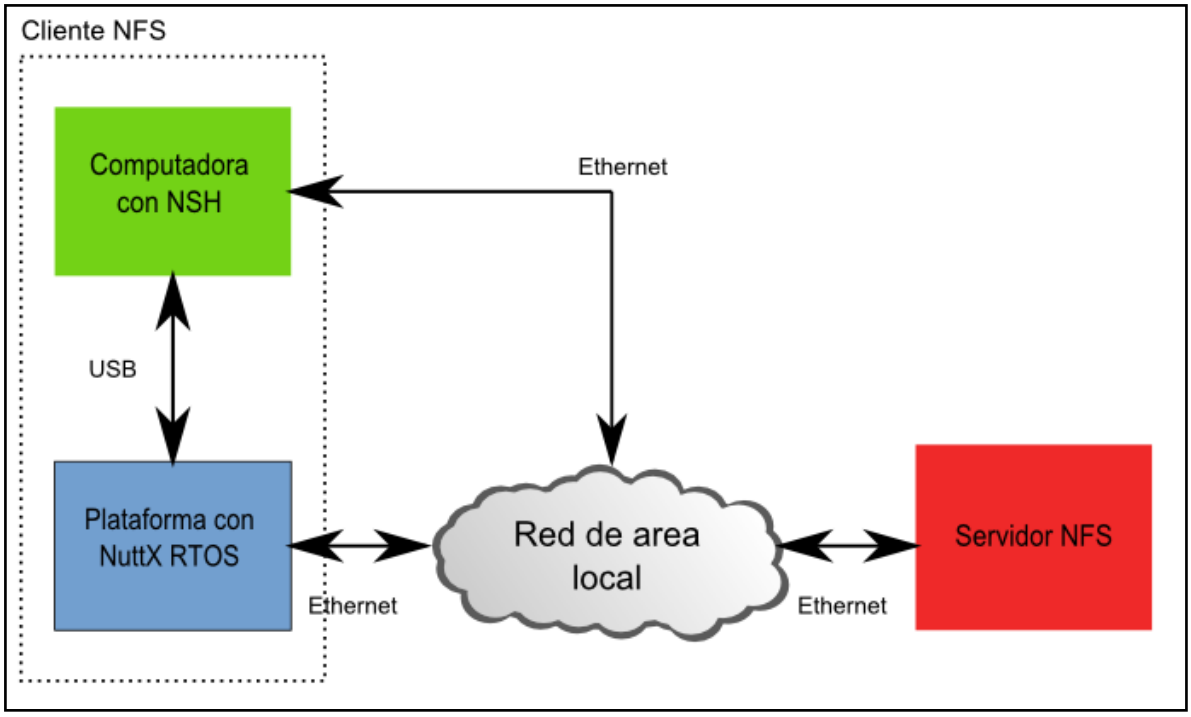


Figura 5.1 Diagrama de bloques del hardware implementado.

5.1.1 Cliente NFS

El cliente NFS fue implementado de manera tal que, se pueda utilizar en cualquiera de las plataformas con soporte para NuttX RTOS. Debido a que el protocolo NFS es un protocolo de red, la plataforma debe poseer un módulo de Ethernet.

Otra de las necesidades que se buscaba solventar con el proyecto, consistió en realizar una optimización en el uso de la memoria al utilizar la nueva función de cliente NFS, por lo que se buscó usar una plataforma con poca memoria, de esta manera, se puede verificar que la función se esté ejecutando correctamente sobre la plataforma. Por lo tanto, se realizaron las pruebas de la nueva función sobre las plataformas TI/Luminary Stellaris LM3S6965 Evaluation Board [4] y EAGLE 100 Single Board [6]. Como se comentó en la sección 1.2, estas tarjetas poseen características similares y a su vez cumplen con la necesidad de poseer un periférico Ethernet que les permite conectarse a la red. Estas tarjetas embebidas también cuentan con un microprocesador basado en el procesador ARM Cortex - M3 [21], el cual posee un alto rendimiento y es especial para implementaciones que requieren un mínimo de memoria. Lo que permitió que estas plataformas sean ideales para realizar una optimización en cuanto al uso racional de la memoria y a su vez comprobar la compatibilidad del cliente NFS en más de una tarjeta.

Para controlar el cliente NFS, se necesita una computadora que posea una terminal de comandos, de manera que se pueda acceder al NSH (Ver sección 3.1.1) ya sea mediante un puerto COM virtual que se obtiene al realizar una conexión USB con las tarjetas o mediante el acceso por medio de telnet al conectar el equipo a la misma red en la que se encuentra la tarjeta que se está utilizando.

5.1.2 Servidor NFS

El servidor NFS es una computadora que posibilita que el cliente NFS, conectado a la misma red, acceda a los ficheros remotos anteriormente configurados en esta. En la implementación del proyecto, se utilizó una computadora con la distribución Ubuntu 10.04 de Linux como sistema operativo. Se configuró el quipo de manera tal que, este compartiera un directorio de forma remota. La configuración de este servidor se encuentra en apéndice A.2, la cual forma parte del tutorial realizado para el sitio web del NuttX RTOS [3].

5.2 Descripción del software

A continuación se detalla el diseño del software que permitió la implementación del cliente NFS.

Como se mencionó con anterioridad, el protocolo NFS y el protocolo RPC son protocolos de red, estos son usados bajo el esquema del Modelo OSI, comentado en la sección 3.3. Por lo que el diseño de la solución se planteó de manera tal que se mantuviera este enfoque en capas.

El diseño general de la solución posee el siguiente esquema:

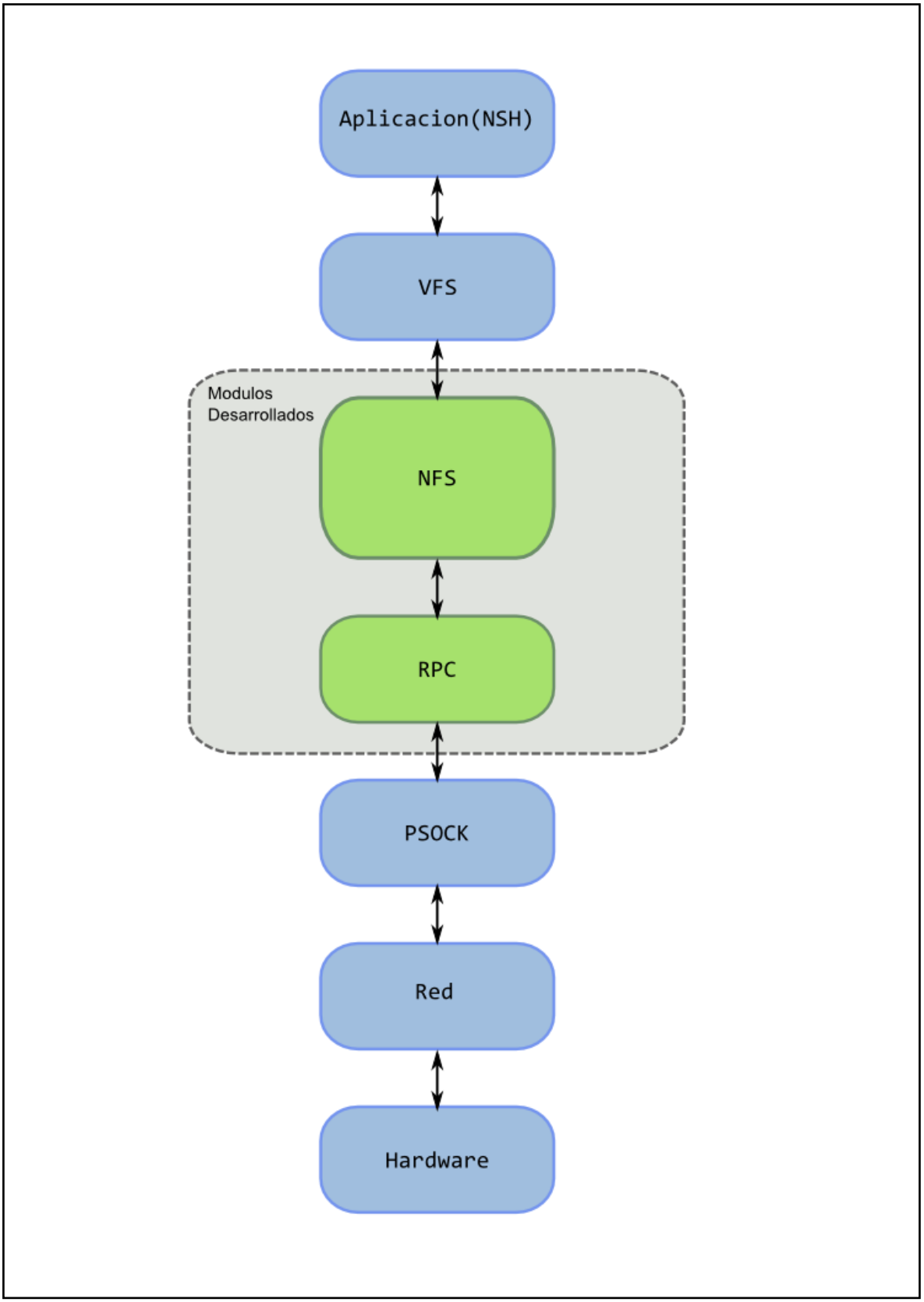


Figura 5.2 Diagrama general de la solución planteada.

A continuación se describen cada uno de los módulos del diagrama general, mostrados en la figura 5.2. Haciendo énfasis en los módulos que fueron diseñados e implementados para crear el cliente NFS.

5.2.1. Aplicación (NSH)

La aplicación que permite el manejo del sistema de archivos de red realizado es NSH, el cual como se comentó en la sección 3.1.1, se trata de un intérprete de comandos para el NuttX RTOS.

Como parte del diseño, se introdujo un nuevo comando de NFS en el intérprete, el cual tiene como función realizar el montaje NFS con el servidor, permitiendo así el acceso de remoto de archivos por parte de cliente.

Este nuevo comando, junto con su respectiva descripción se encuentra en el apéndice A.3, que también forma parte del tutorial realizado para el sitio web de NuttX RTOS [3].

5.2.2. VFS

El NuttX RTOS cuenta con un VFS, el cual facilita la implementación de un sistema de archivos como lo es el protocolo NFS. Este protocolo, como se explicó en la sección 3.3, utiliza “Inodes” o nodos internos para describir el contenido de un sistema de archivos. Por lo tanto, se otorga un “Inode” para el sistema de archivos de red. En el NuttX RTOS, el “Inode” tiene la siguiente estructura:

```
struct inode
{
  FAR struct inode *i_peer;      /* Puntero hacia el mismo nivel del inode */
  FAR struct inode *i_child;     /* Puntero hacia un nivel mas abajo del inode */
  int16_t           i_crefs;     /* Referencias hacia inode */
  uint16_t          i_flags;     /* Flags para inode */
  union inode_ops_u u;           /* Operaciones del Inode  */
  FAR void         *i_private;   /* Datos privados de cada uno de los inodes */
  char              i_name[1];   /* Nombre del inode (variable) */
};
```

Figura 5.3 Estructura de un Inode en NuttX RTOS.

En esta estructura, se hace énfasis en dos variables o estructuras que son vitales para la solución planteada. La primera son los datos privados de cada uno de los nodos *(FAR void *i_private),* que para el caso de NFS se le conoce como la estructura de montaje y la segunda es la estructura que describe las operaciones del Inode (*unión inode_ops_u u),* la cual posee la lista de las operaciones soportadas por el sistema de archivos de red que serán utilizadas por el VFS. Sobre esta última se referirá más adelante en la sección 5.3.3.

La estructura de montaje posee toda la información específica de NFS para realizar un montaje. Esta se le asigna un espacio en memoria cada vez que se hace un montaje de NFS. La estructura de la implementación posee la siguiente forma:

```
struct nfsmount
{
  struct nfsnode  *nm_head;          /* Una lista de archivos abiertos en el punto de montaje */
  sem_t           nm_sem;            /* Usado para asegurar un buen acceso */
  nfsfh_t         nm_fh;             /* Identificador del directorio raiz */
  char            nm_path[NFS_PATHLEN];  /* Camino de directorio ,en el servidor, que es montado */
  struct nfs_fattr nm_fattr;         /* Atributos de archivo nfs*/
  struct rpcclnt  *nm_rpcclnt;       /* RPC estado */
  struct socket   *nm_so;            /* RPC socket */
  struct sockaddr  nm_nam;           /* Direccion del servidor */
  bool            nm_mounted;        /* true: El sistema de archivos esta listo */
  uint8_t         nm_fhsize;         /* Tamaño de la raiz del identificador de archivo */
  uint8_t         nm_sotype;         /* Tipo de socket */
  uint8_t         nm_retry;          /* Maximos intentos */
  uint16_t        nm_timeo;          /* Valor del timeout (en ticks del reloj del sistema) */
  uint16_t        nm_rsize;          /* Tamaño maximo del read RPC */
  uint16_t        nm_wsize;          /* Tamaño maximo del write RPC */
  uint16_t        nm_readdirsize;    /* Tamaño del readdir RPC */
  uint16_t        nm_buflen;         /* Tamaño del I/O buffer */

  /* Union en el stack que se utiliza para llevar el mensaje de llamada mas largo.  NOTA
   * Es usado para el mensaje de respuesta en el caso de WRITE.
   */

  union
  {
    struct rpc_call_pmap    pmap;
    struct rpc_call_mount   mountd;
    struct rpc_call_create  create;
    struct rpc_call_lookup  lookup;
    struct rpc_call_read    read;
    struct rpc_call_remove  removef;
    struct rpc_call_rename  renamef;
    struct rpc_call_mkdir   mkdir;
    struct rpc_call_rmdir   rmdir;
    struct rpc_call_readdir readdir;
    struct rpc_call_fs      fsstat;
    struct rpc_call_setattr setattr;
    struct rpc_call_fs      fs;
    struct rpc_reply_write  write;
  } nm_msgbuffer;

  /* I/O buffer.  Este buffer es usado por todos los mensajes de respuesta excepto
   * para WRITE RPC. En ese caso, es usado para el mensaje de llamada de WRITE que
   * contiene los datos para ser escritos.
   */

  uint32_t        nm_iobuffer[1];    /* El tamaño actual es dado por nm_buflen */
};
```

Figura 5.4 Estructura de montaje para un montaje NFS en NuttX RTOS.

Además, el NuttX RTOS cuenta con otra estructura que es utilizada cada vez que se abre o se encuentra activo un archivo. Esto debido a que el RTOS se diseñó para que trabaje con descriptores de archivos, los cuales usan esta estructura para separar los archivos. Esta tiene la forma mostrada a continuación:

```
struct file
{
  int              f_oflags; /* Flags del modo abierto */
  off_t            f_pos;    /* Position del archivo */
  FAR struct inode *f_inode; /* Interfaz del Driver */
  void             *f_priv;  /* Datos privados por cada archivo */
};
```

Figura 5.5 Estructura de un archivo abierto en NuttX RTOS.

Igualmente, se realizó una estructura específica para el protocolo NFS que contenga toda la información del archivo NFS. A este se le asigna un espacio en memoria cada vez que hay un archivo NFS activo. La estructura es la siguiente:

```
struct nfsnode
{
  struct nfsnode    *n_next;      /* Retained in a singly linked list. */
  uint8_t           n_type;       /* Tipo del archivo */
  uint8_t           n_fhsize;     /* Tamaño en bytes del identificador del archivo */
  uint8_t           n_flags;      /* Flags de nodo */
  struct timespec   n_mtime;      /* Tiempo de la modificacion del archivo  */
  time_t            n_ctime;      /* Tiempo de la creacion del archivo  */
  nfsfh_t           n_fhandle;    /* Identificador del archivo NFS */
  uint64_t          n_size;       /* Tamaño actual del archivo  */
};
```

Figura 5.6 Estructura de un archivo de NFS activo en NuttX RTOS.

Esta, a su vez es almacenada en la estructura de montaje *nfsmount* para llevar una lista de archivos activos durante el montaje.

5.2.3. NFS

El protocolo NFS implementado corresponde a la versión 3, la cual posee 22 procedimientos o rutinas que permiten el manejo del sistema de archivos. Debido a que el NuttX RTOS aún se encuentra en desarrollo, algunos de estos procedimientos quedaron por fuera de la implementación ya que son incompatibles con el sistema operativo. Por tanto se crearon las operaciones del protocolo que concuerdan con las ya existentes en el sistema operativo.

Como se comentó en la sección anterior, en el “Inode” del VFS existe una estructura que contiene todas las funciones para el nodo interno soportadas por NuttX RTOS (*union inode_ops_u u*). Para el caso de NFS, el “Inode” hace referencia a un punto de montaje, por lo que las operaciones se adjuntan en forma de una estructura de operaciones de montaje del protocolo NFS, tal como se muestra en la Figura 5.7.

```
const struct mountpt_operations nfs_operations =
{
  nfs_open,                     /* open */
  NULL                          /* close */
  nfs_read,                     /* read */
  nfs_write,                    /* write */
  NULL,                         /* seek */
  NULL,                         /* ioctl */


  nfs_opendir,                  /* opendir */
  NULL,                         /* closedir */
  nfs_readdir,                  /* readdir */
  NULL,                         /* rewinddir */

  nfs_bind,                     /* bind */
  nfs_unbind,                   /* unbind */
  nfs_statfs,                   /* statfs */

  nfs_remove,                   /* unlink */
  nfs_mkdir,                    /* mkdir */
  nfs_rmdir,                    /* rmdir */
  nfs_rename,                   /* rename */
  NULL                          /* stat */
};
```

Figura 5.7 Estructura de las operaciones de NFS compatibles en NuttX RTOS.

Una vez realizado el montaje de un directorio mediante NFS con el servidor se puede ejecutar distintas operaciones, por ejemplo, ejecutar el comando “mkdir” en el intérprete de funciones NSH. El intérprete llama al VFS para comprobar si la operación existe en el “Inode” del sistema de archivos de red. Si es así, se ejecuta esta función con la indicada en la estructura, el cual en este caso corresponde a “*nfs_mkdir*”.

Cada procedimiento del protocolo NFS utiliza distintas estructuras con un formato específico de envió y recepción de mensajes. Estas estructuras se encuentran especificadas en documento RFC 1813, el cual contiene todas las especificaciones del protocolo NFSv3 [20].

A continuación se describen cada una de las operaciones implementadas del protocolo NFSv3 para el VFS del NuttX RTOS:

5.2.3.1 NFS_OPEN

El protocolo NFS no posee ningún procedimiento que ejecute esta operación. Sin embargo, como se comentó en la sección 5.3.2, esta función es necesaria ya que el NuttX RTOS lleva una tabla de descriptores con todos los archivos que se encuentran abiertos.

En el protocolo NFSv3 se encuentra la función “CREATE”. Esta función en NuttX RTOS es realizada por la función “OPEN” e indicada por las banderas *oflags* de los parámetros de entrada al valor O_CREAT. Por lo que se optó por implementar estas dos operaciones en una misma.

La función de NFS_OPEN recibe cuatro parámetros de entrada:

- *FAR struct file *filep*: Estructura que contiene el archivo abierto o activo especifica de NuttX RTOS.
- *FAR const char *relpath*: Ruta del archivo a abrir.
- *int oflags*: Banderas de los privilegios del archivo.
- *mode_t mode*: Banderas que contienen permisiones correspondientes del archivo.

La salida de esta operación, retorna un valor de cero si se realizó correctamente, por otro lado, si retorna un valor negativo, indica un error.

La figura muestra el diagrama de flujo de la operación NFS_OPEN:

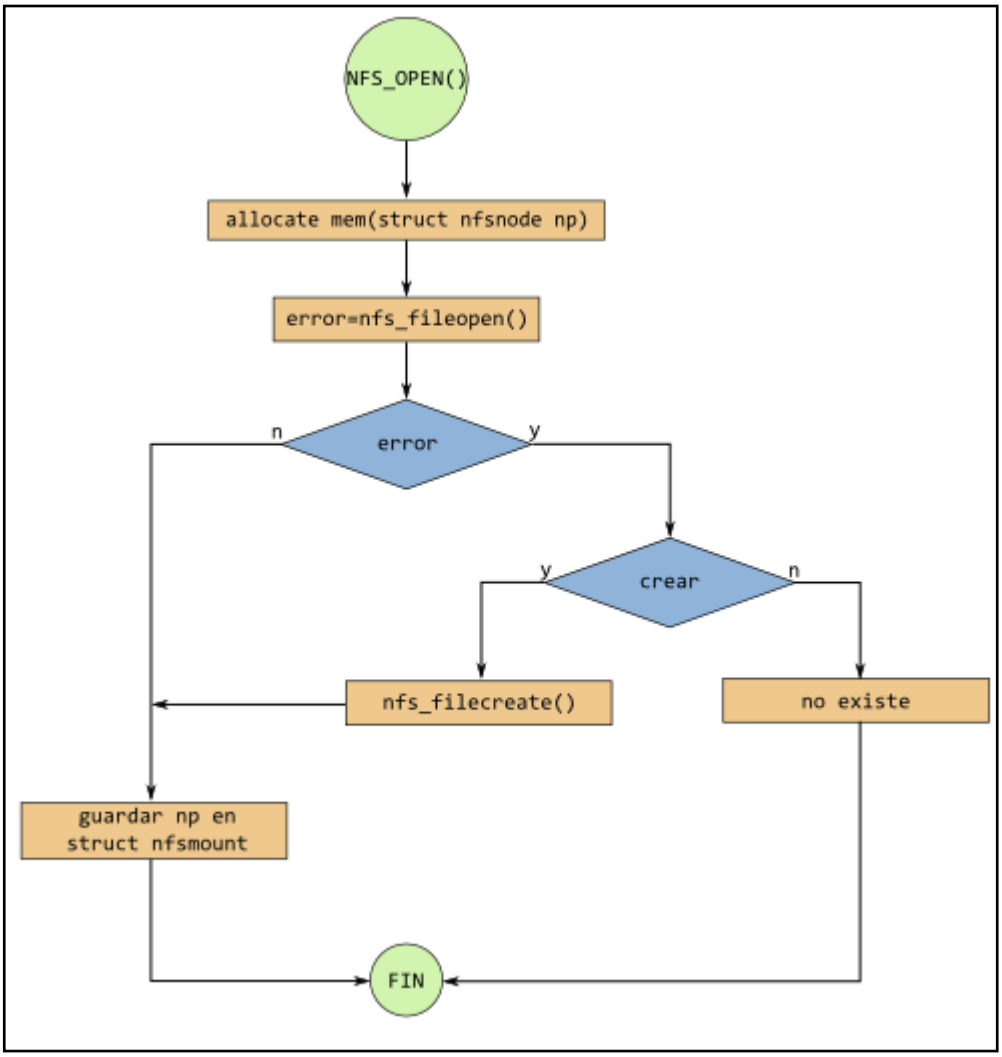


Figura 5.8 Diagrama de flujo de la operación NFS_OPEN en NuttX RTOS.

Inicialmente, esta operación realiza una pre-asignación de los datos del archivo privado para describir este mismo. Posteriormente, trata de abrir el archivo mediante la función *nfs_fileopen(),* si no se presenta ningún error al retorno de la función, se guarda esta estructura de datos privados modificada en el punto de montaje y en la instancia del archivo abierto. Por el contrario si existe un error, se verifica si las banderas *oflags* son iguales al valor O_CREATE, y de serlo se procede a llamar la función *nfs_filecreate()* para crear el archivo. De otra manera se indica que el archivo no existe.

Las funciones *nfs_fileopen()* y *nfs_filecreate()* posee lo mismos parámetros de entrada y salida que NFS_OPEN. Sus diagramas de flujo se muestran a continuación:

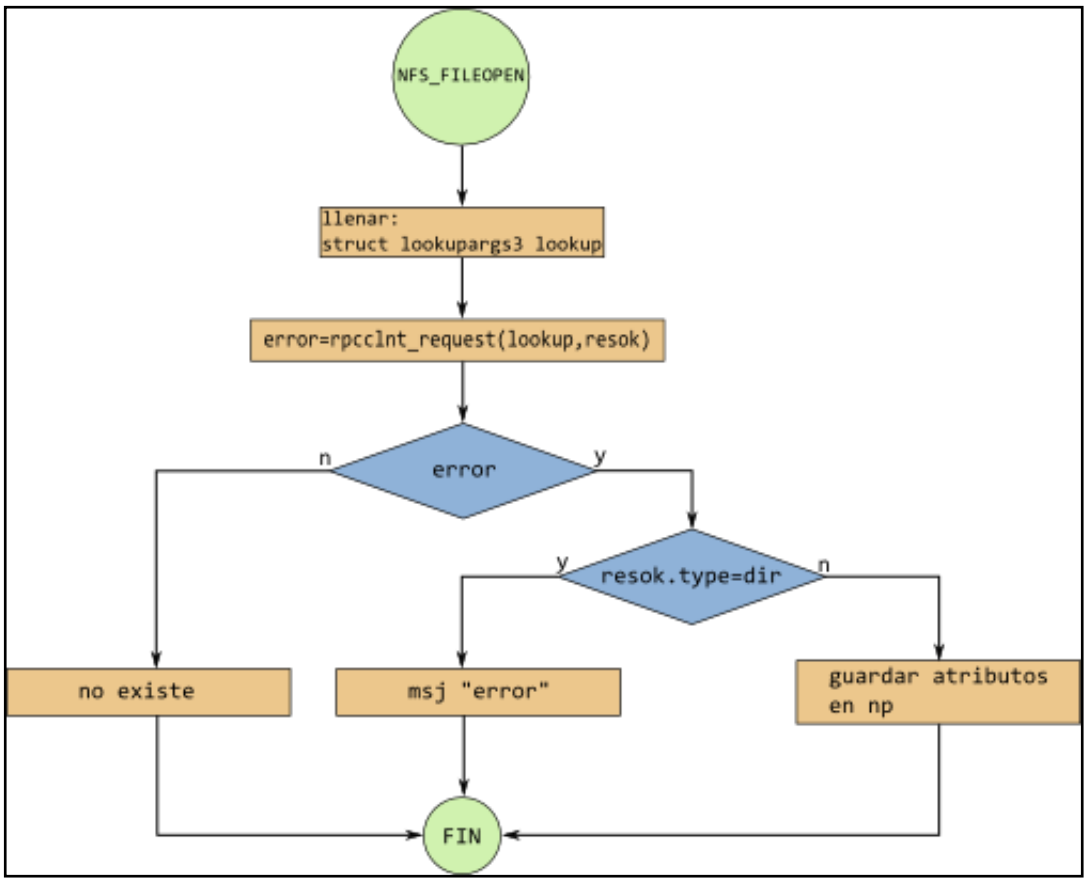


Figura 5.9 Diagrama de flujo de función *nfs_fileopen*() usada en NFS_OPEN.

Inicialmente, esta función envía una petición RPC que lleva la información del procedimiento de LOOKUP del protocolo NFS, para comprobar si en el servidor existe el archivo o no. Si no existe, retorna un valor de error, de lo contrario verifica en el mensaje de respuesta (llamado *Resok*), enviado por el servidor, si se trata de un directorio o de un archivo. Si es un directorio, retorna un error ya que la función es solo para archivos, en caso contrario guarda los atributos del archivo contenidos en *Resok* dentro de la estructura *nfsnode* de la instancia.

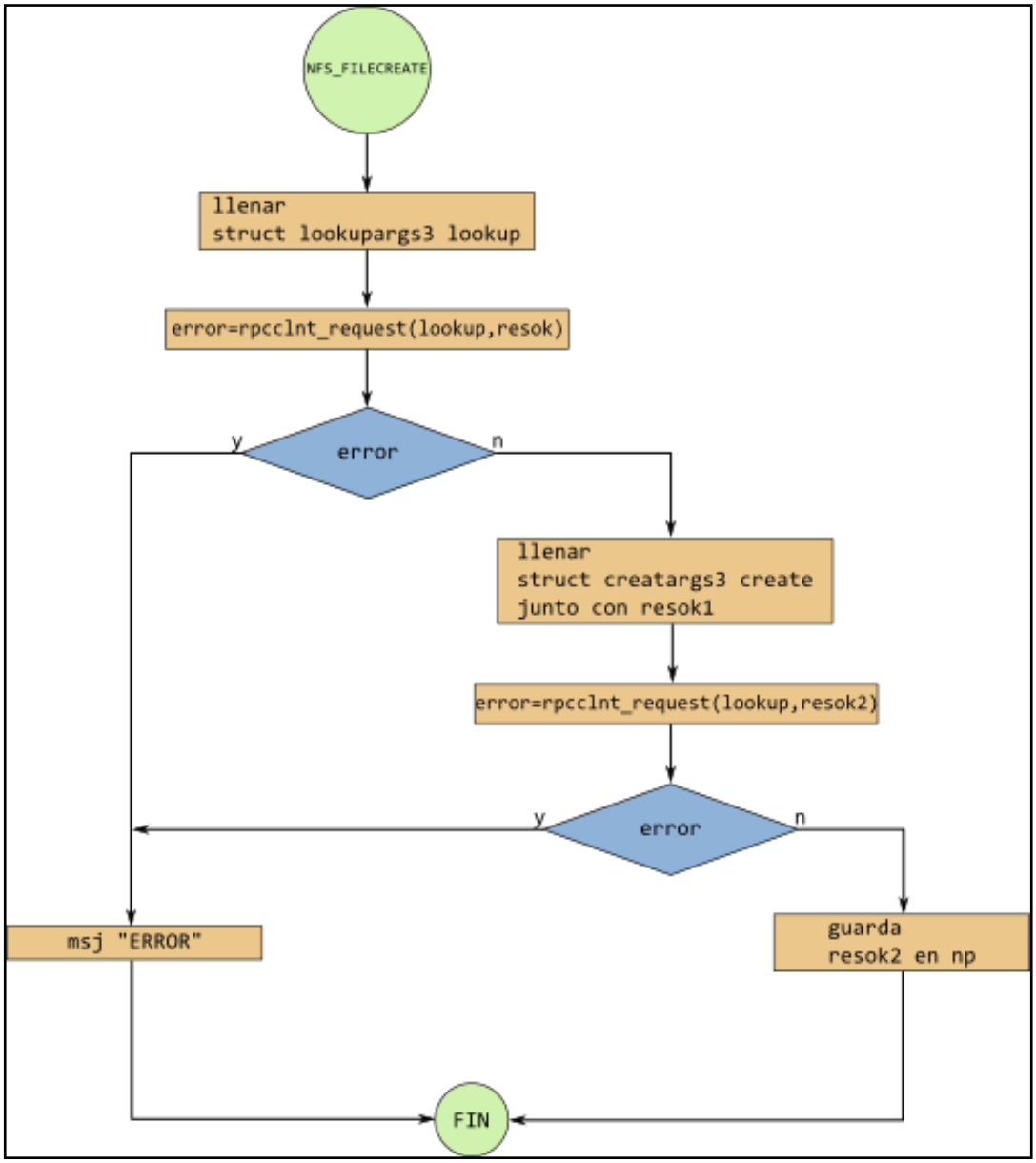


Figura 5.10 Diagrama de flujo de función *nfs_filecreate()* usada en NFS_OPEN.

Al inicio esta función envía una petición RPC que contiene la información del procedimiento de LOOKUP del protocolo NFS, para comprobar si en el servidor existe el directorio donde se desea crear el archivo. Si no existe, retorna un valor de error, en caso contrario, se envía otra petición de RPC con el procedimiento de CREATE del protocolo NFS, conteniendo todos los atributos con los que se van a crear el archivo. Si existe algún problema en la creación del archivo, se retorna un error. Sin embargo, si el archivo se crea con éxito, se

guardan los atributos del archivo que se encuentra contenidos en la respuesta *“Resok*”, en la estructura *nfsnode* de la instancia.

5.2.3.2 NFS_READ

La función de NFS_READ recibe tres parámetros de entrada:

- *FAR struct file *filep*: Estructura que contiene el archivo abierto o activo especifica de NuttX RTOS.
- *char *buffer*: Buffer en el cual se guarda los datos leídos.
- *size_t buflen*: Tamaño del buffer.

La salida de esta operación, si se realiza correctamente, retorna valores positivos indicando el número de Bytes leídos. En caso contrario, retorna un valor negativo indicando el error ocurrido.

La figura muestra el diagrama de flujo de la operación NFS_READ:

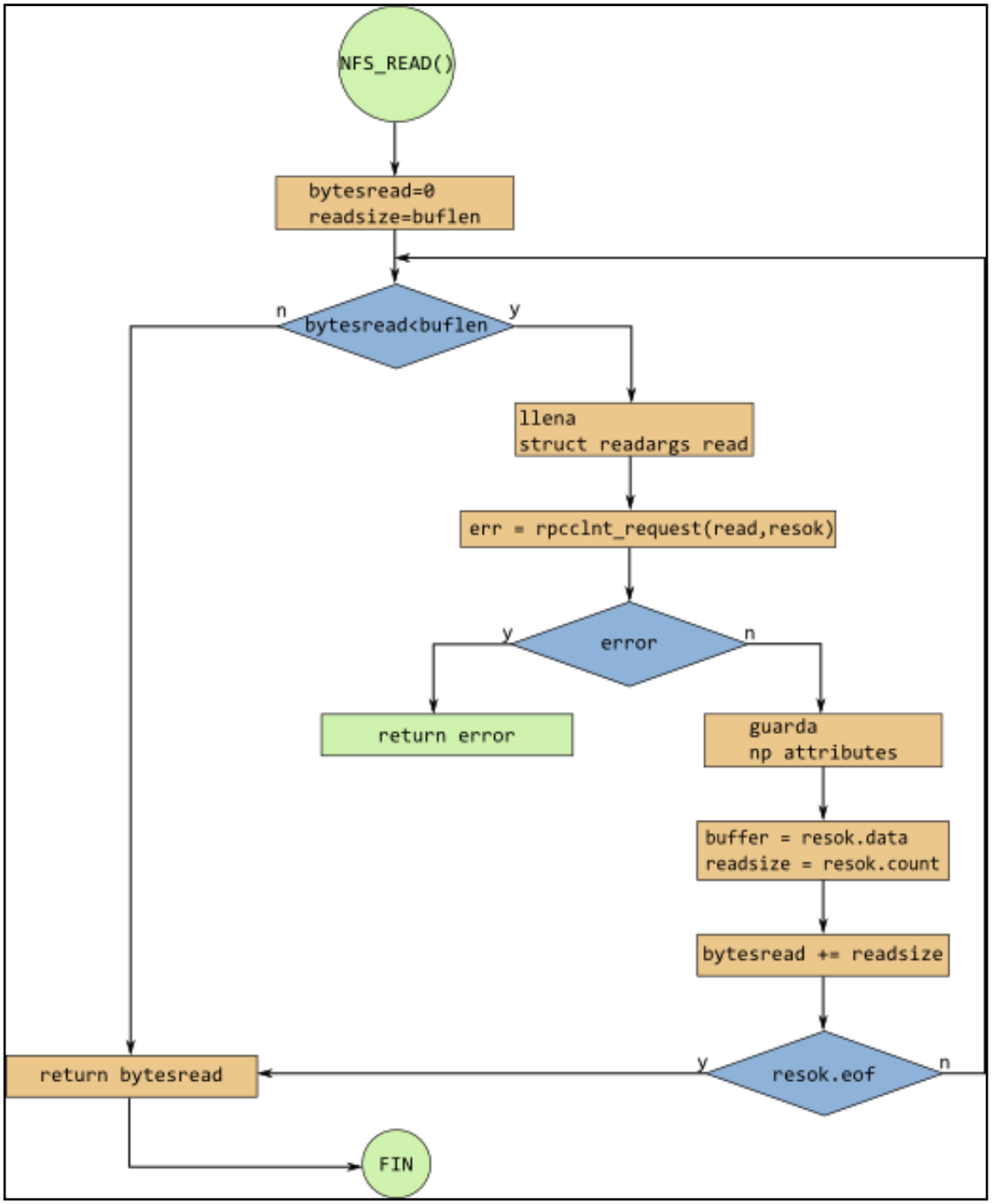


Figura 5.11 Diagrama de flujo de la operación NFS_READ en NuttX RTOS.

.

Esta función se encuentra realizando un ciclo constante de llenado del buffer, al finalizar se retorna un valor indicando el total de Bytes leídos (*bytesread*). Dentro del ciclo, se envía una petición RPC que contiene la información del procedimiento de READ del protocolo NFS, si la petición es exitosa se procede a guardar los atributos del archivo contenidos en la respuesta "*Resok*", en la estructura *nfsnode* de la instancia. Además, se cargan los datos

leídos del servidor al buffer del usuario y se le agrega el tamaño de esta lectura a la variable “*bytesread*”. Si no se encuentra en el final del archivo, se vuelve a realizar ciclo.

5.2.3.3 NFS_WRITE

La operación NFS_WRITE es similar a la operación de NFS_READ, la cual posee los mismos tres parámetros de entrada. Esta rutina retorna el valor de los bytes escritos si la operación se realiza con éxito (*writesize*). A diferencia de la rutina de lectura, el buffer contiene los datos que se escribirán, el cual es enviado como parte de la petición RPC que contiene la información del procedimiento de WRITE del protocolo NFS.

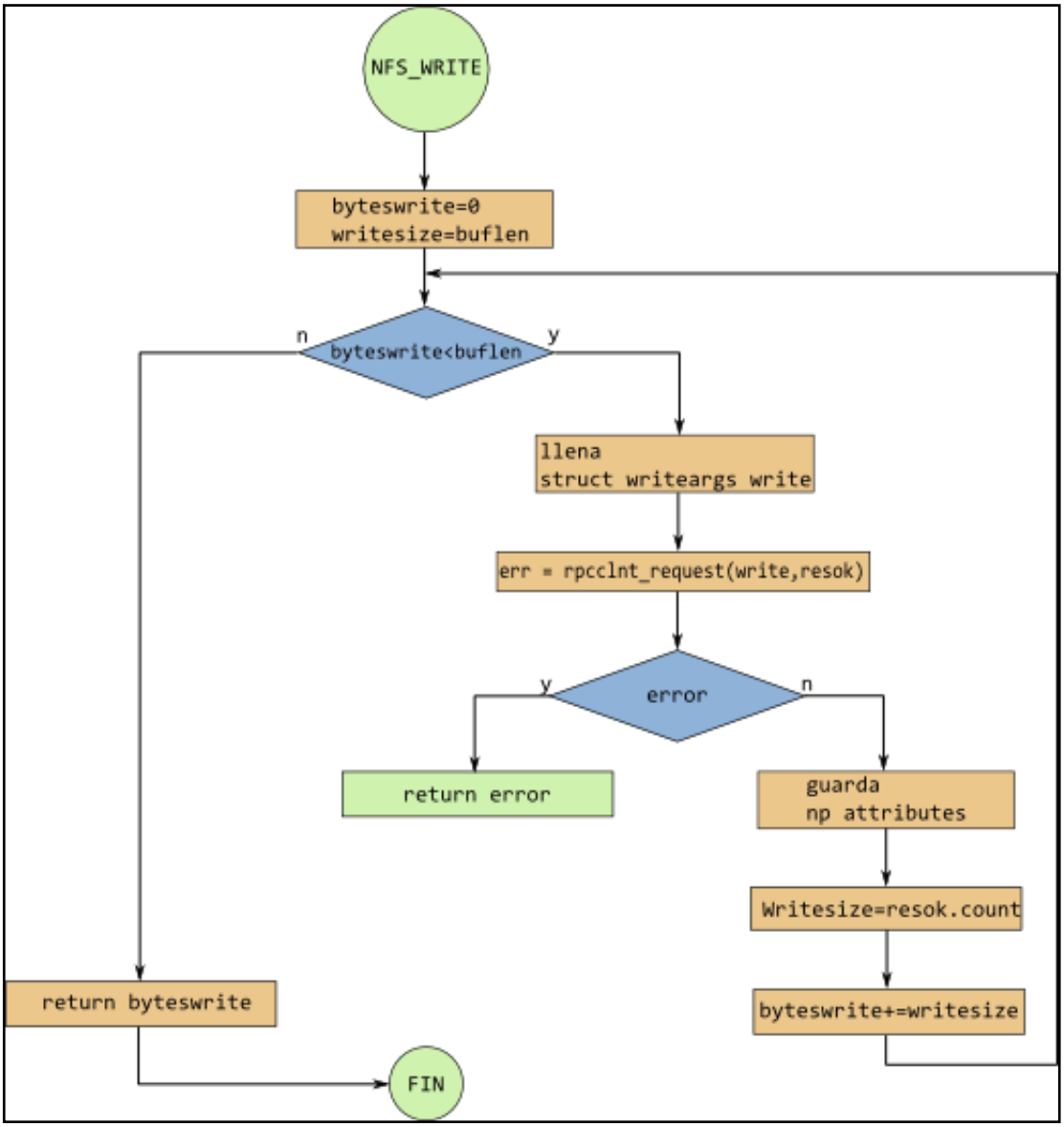


Figura 5.12 Diagrama de flujo de la operación NFS_WRITE en NuttX RTOS.

5.2.3.4 NFS_OPENDIR

La función de NFS_OPENDIR recibe tres parámetros de entrada:

- *struct inode *mountpt*: Estructura que representa un “Inode” en NuttX RTOS.
- *const char *relpath*: Ruta del directorio.
- *struct fs_dirent_s *dir*: Estructura utilizada para abrir un directorio.

La salida de esta operación retorna un valor de cero si se realiza correctamente, al contrario, retorna un valor negativo indicando el error ocurrido.

La figura muestra el diagrama de flujo de la operación NFS_OPENDIR:

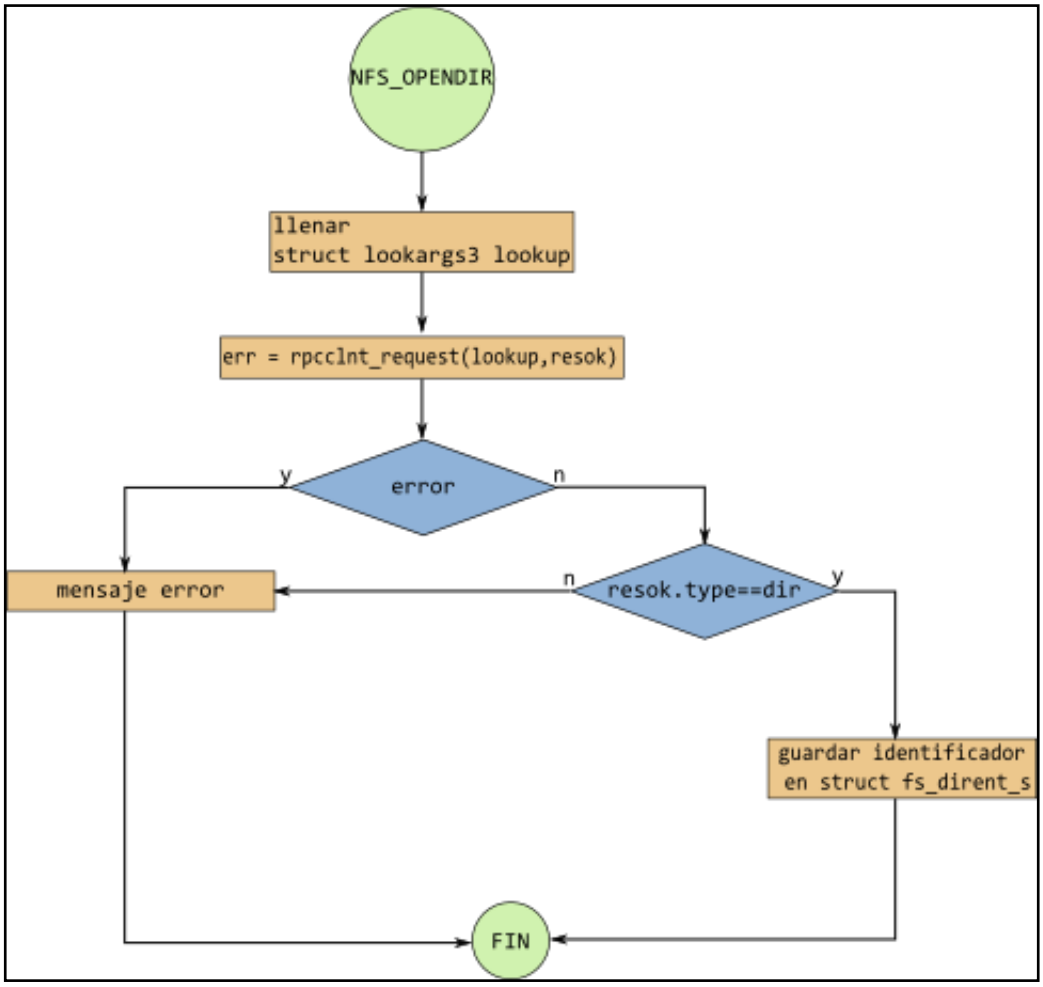


Figura 5.13 Diagrama de flujo de la operación NFS_OPENDIR en NuttX RTOS.

Al comienzo, esta función inicializa la porción especifica de NFS que se encuentra en la estructura *fs_dirent_s,* la cual describe al directorio que se encuentra abriendo. Posteriormente, se envía una petición RPC que contiene la información del procedimiento de LOOKUP del protocolo NFS, para comprobar si en el servidor existe el directorio. De no existir ningún error, se guarda el identificador del directorio en la estructura *fs_dirent_s* de la instancia.

5.2.3.5 NFS_READDIR

La función de NFS_READDIR recibe tres parámetros de entrada:

- *struct inode *mountpt*: Estructura que representa un “Inode” en NuttX RTOS.

- *struct fs_dirent_s *dir*: Estructura utilizada cuando se tiene abierto un directorio.

La salida de esta operación retorna un valor de cero si se realiza correctamente o retorna un valor negativo indicando el error que ocurrido.

La figura muestra el diagrama de flujo de la operación NFS_READDIR:

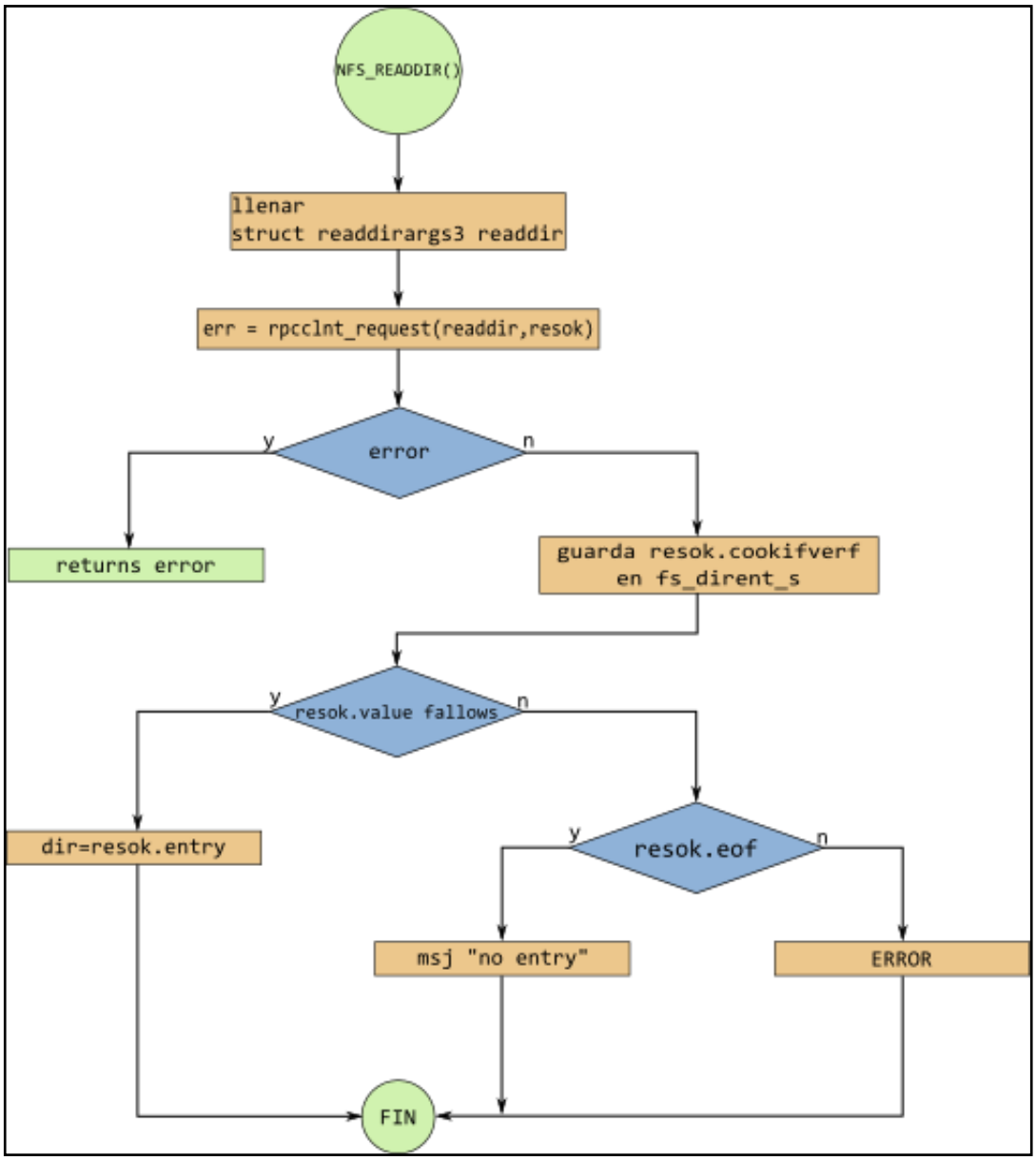


Figura 5.14 Diagrama de flujo de la operación NFS_READDIR en NuttX RTOS.

A través de una petición RPC, esta función envía la información del directorio contenida en la estructura *fs_dirent_s*. La petición contiene la información procedimiento de NFS_READDIR del protocolo NFS. Seguidamente,

se verifica si la variable "*value_fallows*" del mensaje de respuesta (Resok), contiene entradas. Si no tiene entradas, se revisa si es el fin del directorio y se envía un error indicando que no hay entradas.

Si la variable "*value_fallows*" indica que contiene entradas, se procede a guardar la información de la primera entrada en la estructura *fs_dirent_s* de la instancia.

5.2.3.6 NFS_BIND

La función de NFS_BIND recibe 2 parámetros de entrada:

- *FAR const void *data*: Datos aportados desde el comando NFS introducido en NSH.
- *FAR void **handle*: Variable que tiene la referencia de la estructura de montaje *nfsmount*.

A su salida, esta operación retorna un valor de cero si se realizó correctamente o retorna un valor negativo indicando el error ocurrido.

La figura muestra el diagrama de flujo de la operación NFS_ BIND:

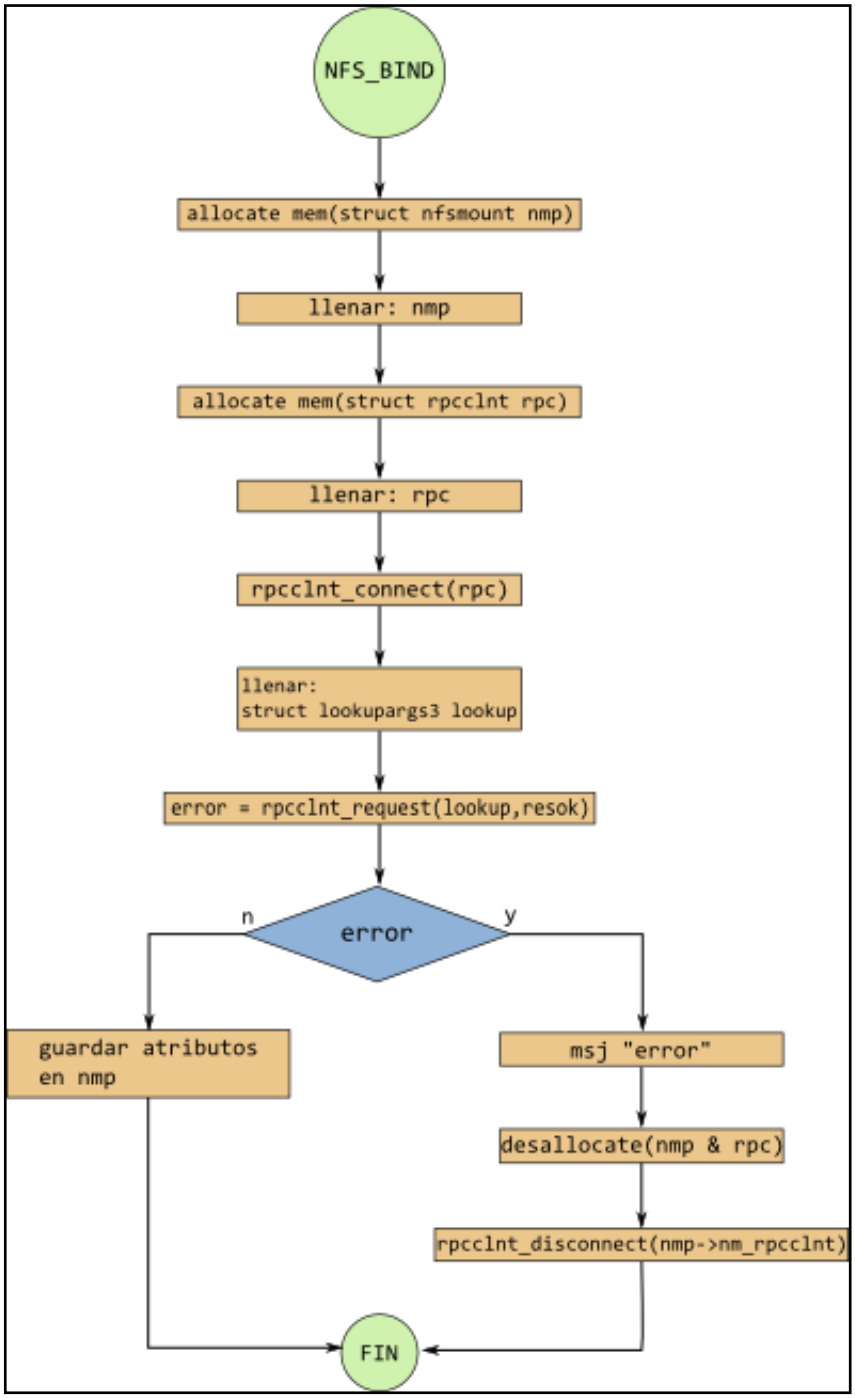


Figura 5.15 Diagrama de flujo de la operación NFS_ BIND en NuttX RTOS.

Esta función es la que se encarga de realizar el montaje NFS. Inicialmente, realiza una pre-asignación de los datos privados de la estructura de montaje *nfsmount*. La variable "*data*" contiene parte de los datos de la estructura, mientras que otros son parte del mismo protocolo NFS. Posteriormente, se realiza una pre-asignación de los datos privados de la estructura *rpcclnt*, la cual contiene el

estado del protocolo RPC. Luego, se ejecuta la función *rpcclnt_connect()* (ver sección 5.3.4) , que permite realizar la conexión correspondiente entre el cliente y el servidor, mediante el uso del protocolo RPC. Finalmente, si la función *rpcclnt_connect()* no presentó ningún tipo de error, se realiza una petición RPC que lleva la información del procedimiento de LOOKUP del protocolo NFS, para comprobar si en el servidor existe el directorio que se ha montado. De no recibir un error en la respuesta, se guardan los atributos faltantes del directorio en la estructura *nfsmount* y además se copia la referencia de esta misma estructura en la variable "*handle*". De lo contrario, se elimina la asignación de memoria de las estructuras *nfsmount* y *rpcclnt,* así como realizar la función de *psock_close()* para ejecutar la desconexión.

5.2.3.7 NFS_UNBIND

La función de NFS_UNBIND recibe un parámetro de entrada:

- *FAR void **handle*: Variable que tiene la referencia de la estructura de montaje *nfsmount*.

La salida de esta operación retorna un valor de cero si se realizó correctamente o retorna un valor negativo indicando el error ocurrido.

La figura muestra el diagrama de flujo de la operación NFS_ UNBIND:

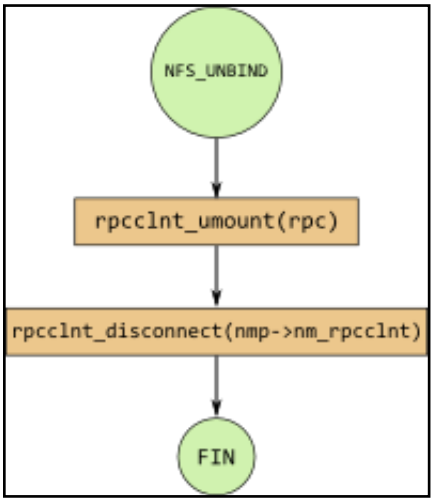


Figura 5.16 Diagrama de flujo de la operación NFS_ UNBIND en NuttX RTOS.

Esta rutina se encarga de llamar a la función *rpcclnt_umount(),* para desmontar el sistema de archivo. Seguidamente, ejecuta la función *psock_close()* para realizar la desconexión del servidor y finalmente libera todas a las estructuras que se asignaron en memoria, de manera que no quede datos inservibles dentro de la memoria.

5.2.3.8 NFS_STATFS

La función de NFS_STATFS recibe dos parámetros de entrada:

- *struct inode *mountpt*: Estructura que representa un “Inode” en NuttX RTOS.
- *FAR struct statfs *sbp:* Estructura que contiene las estadísticas del sistema de archivos.

La salida de esta operación retorna un valor de cero si se realizó correctamente o retorna un valor negativo indicando el error ocurrido. La figura muestra el diagrama de flujo de la operación NFS_STATFS:

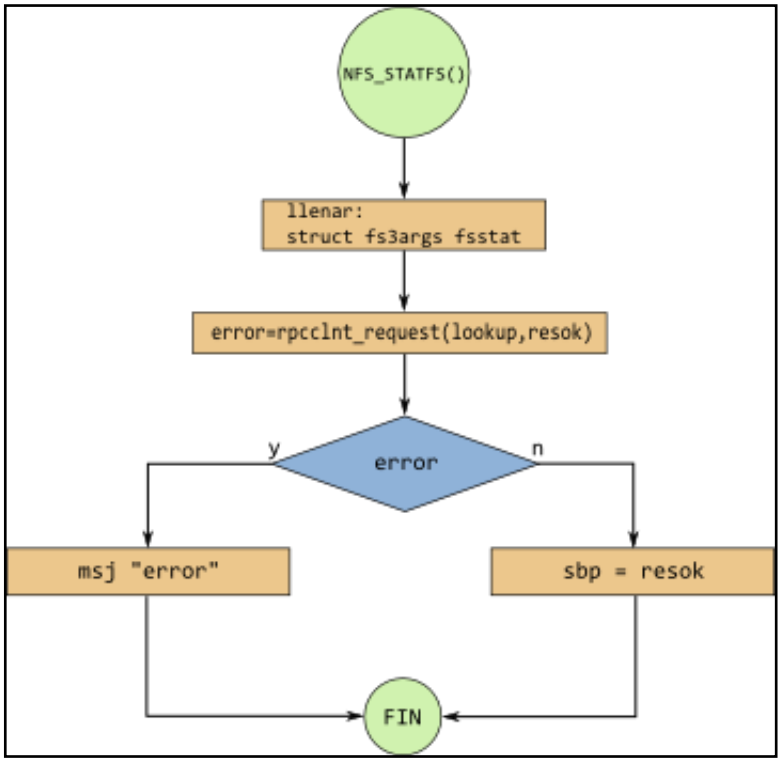


Figura 5.17 Diagrama de flujo de la operación NFS_STATFS en NuttX RTOS.

Esta rutina envía una petición RPC, la cual contiene información del procedimiento de FSSTAT del protocolo NFS. Si la petición es satisfactoria, se pasan las estadísticas enviadas por el servidor a la estructura *statfs.*

5.2.3.9 NFS_REMOVE

La función de NFS_REMOVE recibe dos parámetros de entrada:

- *struct inode *mountpt*: Estructura que representa un “Inode” en NuttX RTOS.
- *const char *relpath:* Trayectoria del archivo.

La salida de esta operación retorna un valor de cero si se realizó correctamente o retorna un valor negativo indicando el error ocurrido. La figura muestra el diagrama de flujo de la operación NFS_REMOVE:

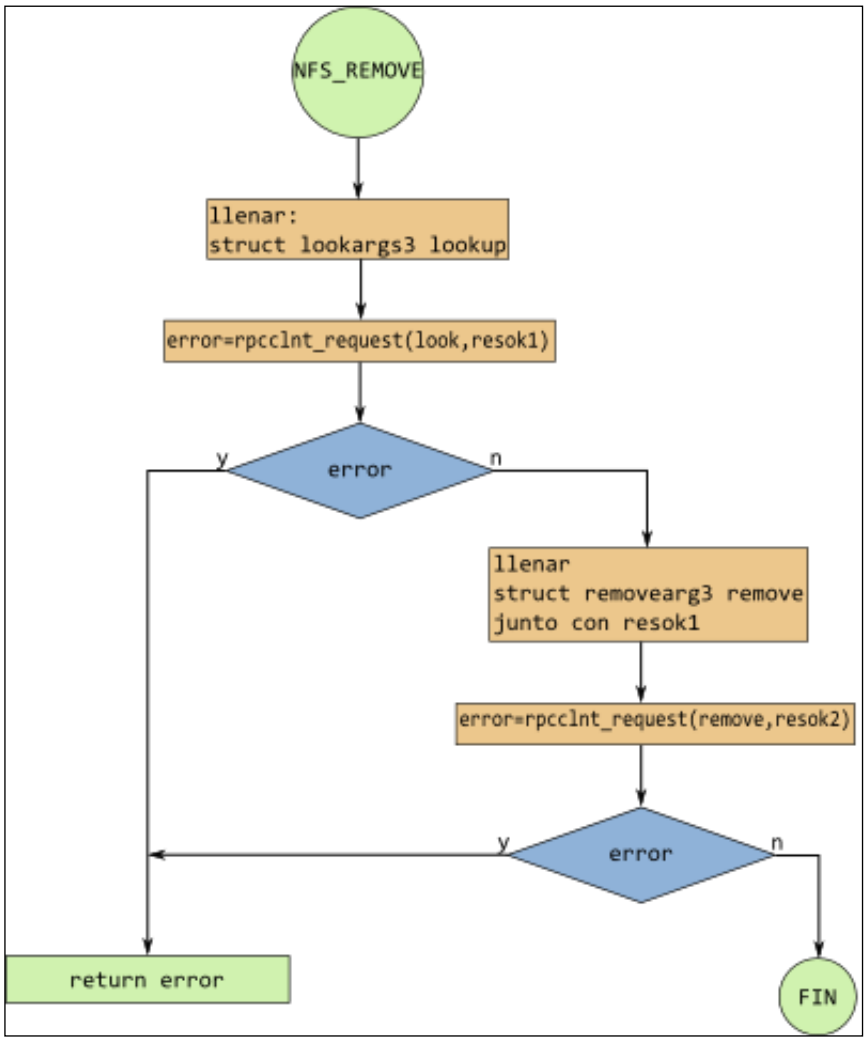


Figura 5.18 Diagrama de flujo de la operación NFS_REMOVE en NuttX RTOS.

Inicialmente, esta función envía una petición RPC que contiene la información del procedimiento de LOOKUP del protocolo NFS, para comprobar si en el servidor existe el archivo o no. Si no existe, retorna un valor de error, de lo contrario, se envía otra petición de RPC con el procedimiento de REMOVE del protocolo NFS el cual contiene identificador del archivo que permite remover el archivo.

5.2.3.10 NFS_MKDIR

La función de NFS_MKDIR recibe dos parámetros de entrada:

- *struct inode *mountpt*: Estructura que representa un “Inode” en NuttX RTOS.
- *const char *relpath:* Trayectoria del archivo.
- *mode_t mode*: Banderas que contienen permisiones correspondientes del directorio.

La salida de esta operación retorna un valor de cero si se realizó correctamente o retorna un valor negativo indicando el error ocurrido. La figura muestra el diagrama de flujo de la operación NFS_ MKDIR:

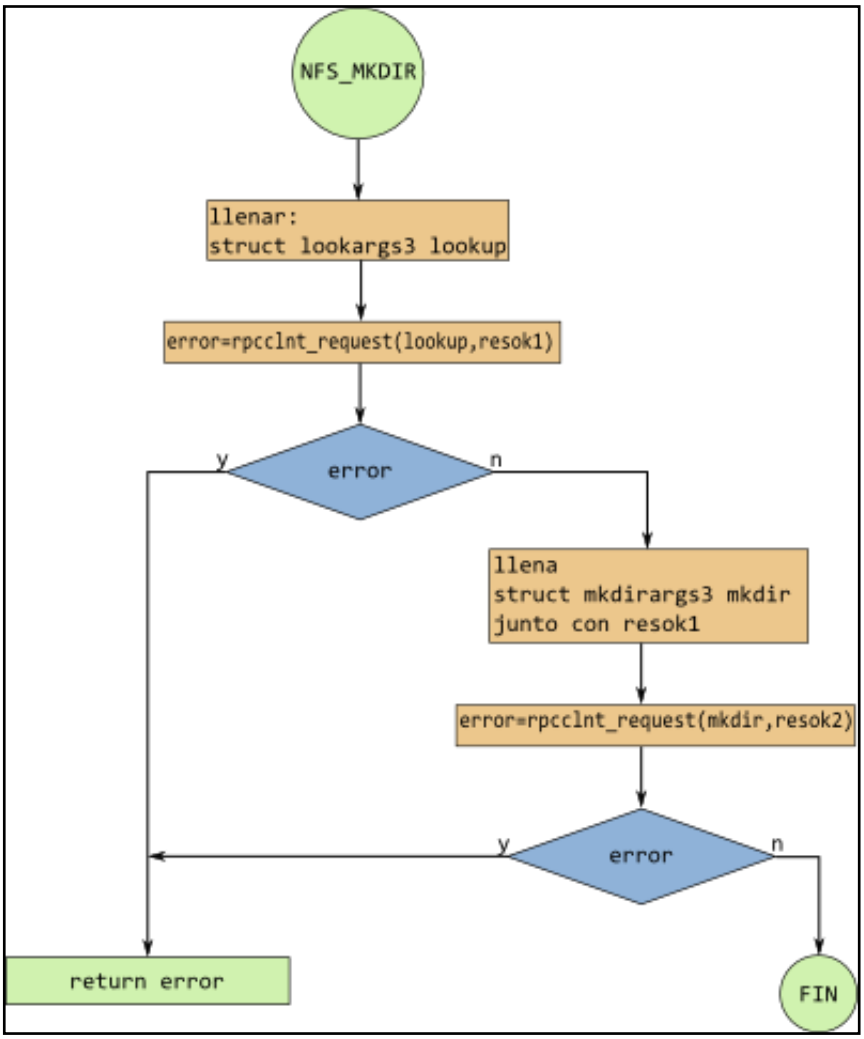


Figura 5.19 Diagrama de flujo de la operación NFS_ MKDIR en NuttX RTOS.

Esta rutina envía una petición RPC que contiene la información del procedimiento de LOOKUP del protocolo NFS, para comprobar si en el servidor

existe el directorio donde se desea crear el nuevo directorio. Si no existe, retorna un valor de error pero de lo contrario se envía otra petición de RPC con el procedimiento de MKDIR del protocolo NFS conteniendo los atributos del nuevo directorio.

5.2.3.11 NFS_RMDIR

La función de NFS_RMDIR recibe dos parámetros de entrada:

- *struct inode *mountpt*: Estructura que representa un “Inode” en NuttX RTOS.
- *const char *relpath:* Trayectoria del directorio.

La salida de esta operación retorna un valor de cero si se realizó correctamente o retorna un valor negativo indicando el error ocurrido. La figura muestra el diagrama de flujo de la operación NFS_ RMDIR:

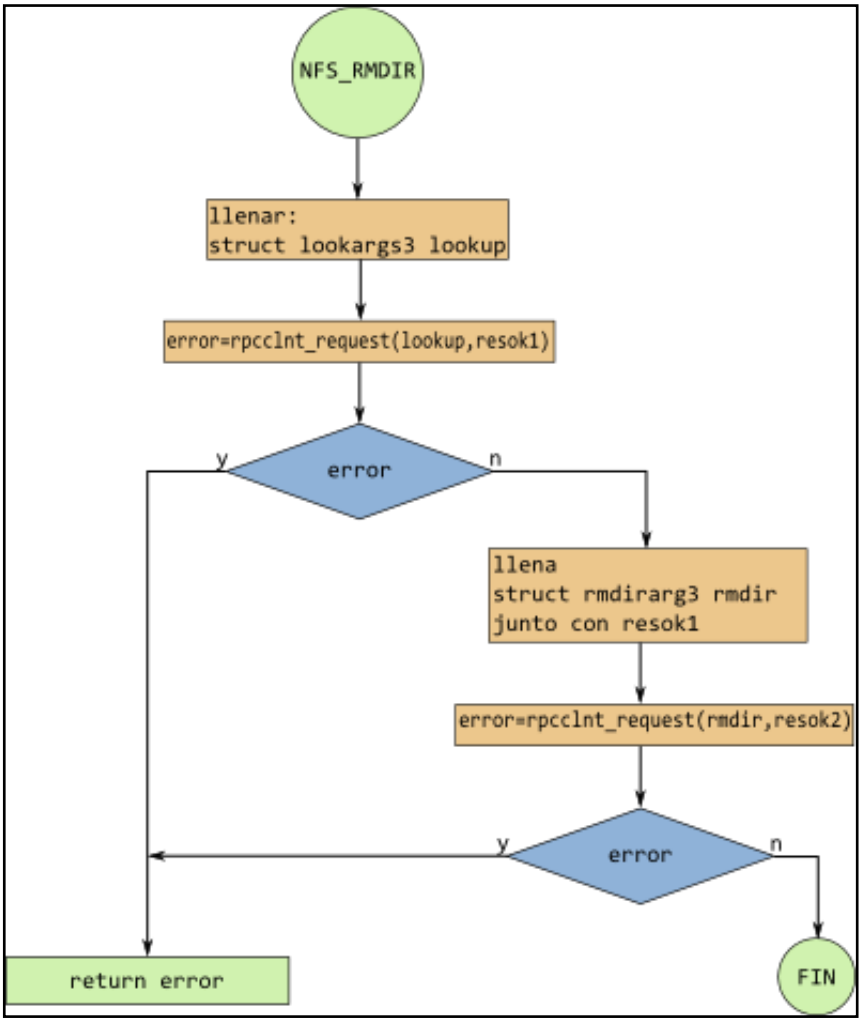


Figura 5.20 Diagrama de flujo de la operación NFS_ RMDIR en NuttX RTOS.

Esta rutina envía una petición RPC que contiene la información del procedimiento de LOOKUP del protocolo NFS, para comprobar si en el servidor existe el directorio o no. Si no existe, retorna un valor de error pero de lo contrario se envía otra petición de RPC con el procedimiento de RMDIR del protocolo NFS conteniendo el identificador del directorio que permite remover mismo.

5.2.3.12 NFS_RENAME

La función de NFS_RENAME recibe tres parámetros de entrada:

- *struct inode *mountpt*: Estructura que representa un “Inode” en NuttX RTOS.
- *const char *oldrelpath:* Trayectoria vieja del archivo.

- *const char *newrelpath:* Trayectoria nueva del archivo.

La salida de esta operación se retorna un valor de cero si esta se realizó correctamente o se retorna un valor negativo que indica el error ocurrido. La figura muestra el diagrama de flujo de la operación NFS_ RENAME:

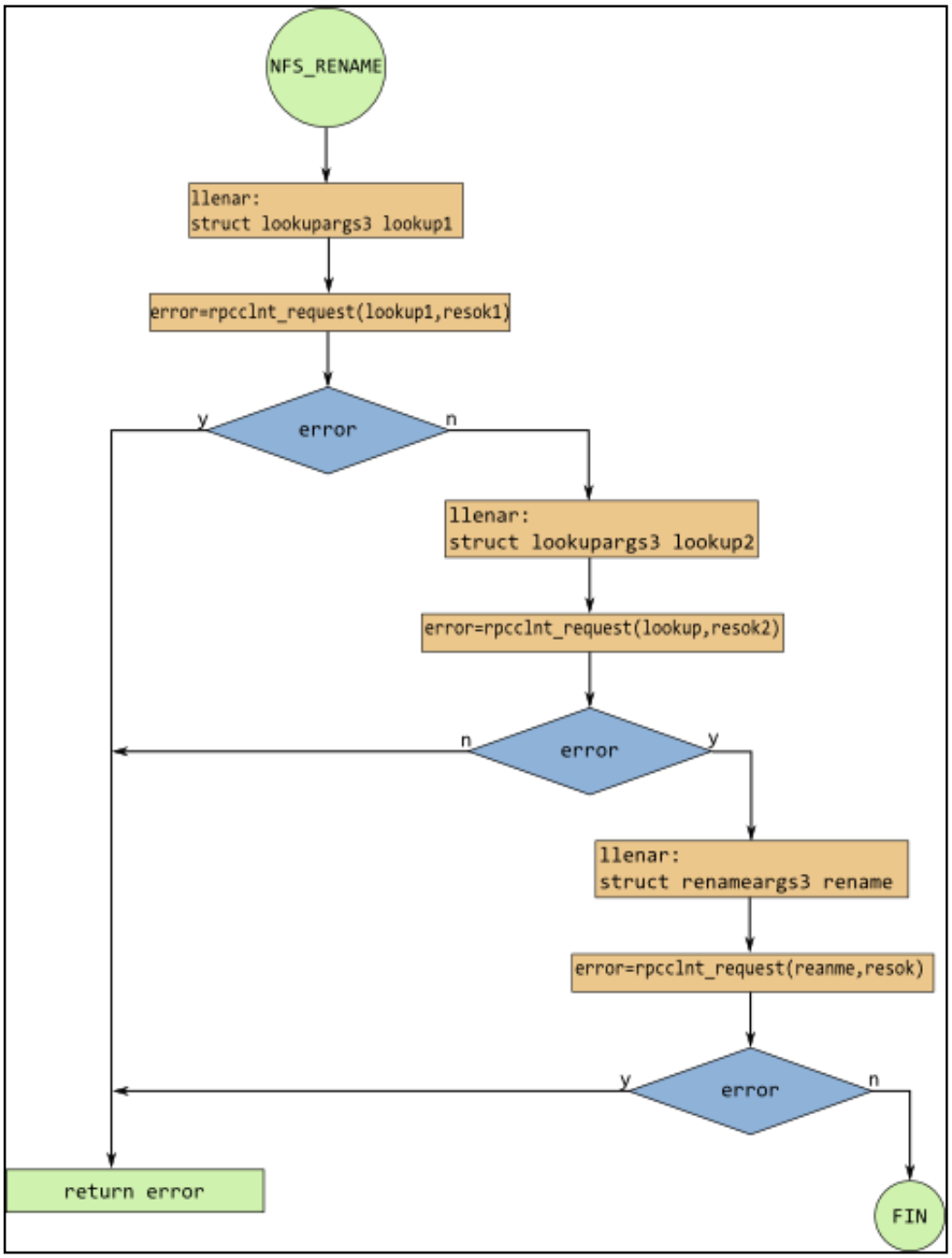


Figura 5.21 Diagrama de flujo de la operación NFS_ RENAME en NuttX RTOS.

Esta función envía una petición RPC, la cual contiene la información del procedimiento de LOOKUP del protocolo NFS, para comprobar si en el servidor existe el archivo que se desea renombrar. Si no existe, retorna un valor de error. En caso contrario, se envía otra petición de RPC del procedimiento de LOOKUP, con una rutina para verificar la existencia de algún archivo con el nombre que se desea renombrar el otro archivo. De no existir, se envía otra petición de RPC con el procedimiento de RENAME del protocolo NFS, ambos conteniendo nombres para que se ejecute el renombramiento del archivo inicial.

5.2.4 RPC

Como se menciona en la sección 3.1.3, este protocolo se encarga de mantener la sesión con el servidor a través de la red, de esta manera, no se requiere trabajar directamente con las tecnologías de capa de red.

Al igual que el NFS, el RPC posee estructuras con un formato específico de envio y recibimiento de mensajes. Estas estructuras están especificadas en documento RFC 1831 que contiene todas las especificaciones del protocolo RPCv2 [22].

Se utilizó este protocolo para implementar la solución del cliente NFS. Este protocolo encapsula los datos dados por NFS para su envío y desencapsula los datos a la hora de la recepción.

Las funciones realizadas del protocolo RPC se describen a continuación:

5.2.4.1 RPCCLNT_REQUEST

Esta función se encarga de realizar las peticiones del protocolo RPC.

La función de RPCCLNT_REQUEST recibe ocho parámetros de entrada:

- *FAR struct rpcclnt *rpc*: Estructura que contiene el estado de RPC en NuttX RTOS.

- *int procnum:* Numero del proceso a solicitar.
- *int prog:* El número del programa que se solicita.
- *int version:* Numero de versión del programa.
- *FAR void *request:* Esta variable contiene la estructura del mensaje de llamada RPC.
- *size_t reqlen:* Tamaño del mensaje de llamada.
- *FAR void *response:* Contiene la estructura el mensaje de respuesta RPC.
- *size_t resplen:* Tamaño del mensaje de respuesta.

La salida de esta operación retorna un valor de cero si se realiza correctamente o retorna un valor negativo indicando el error ocurrido. La figura muestra el diagrama de flujo de la operación RPCCLNT_REQUEST:

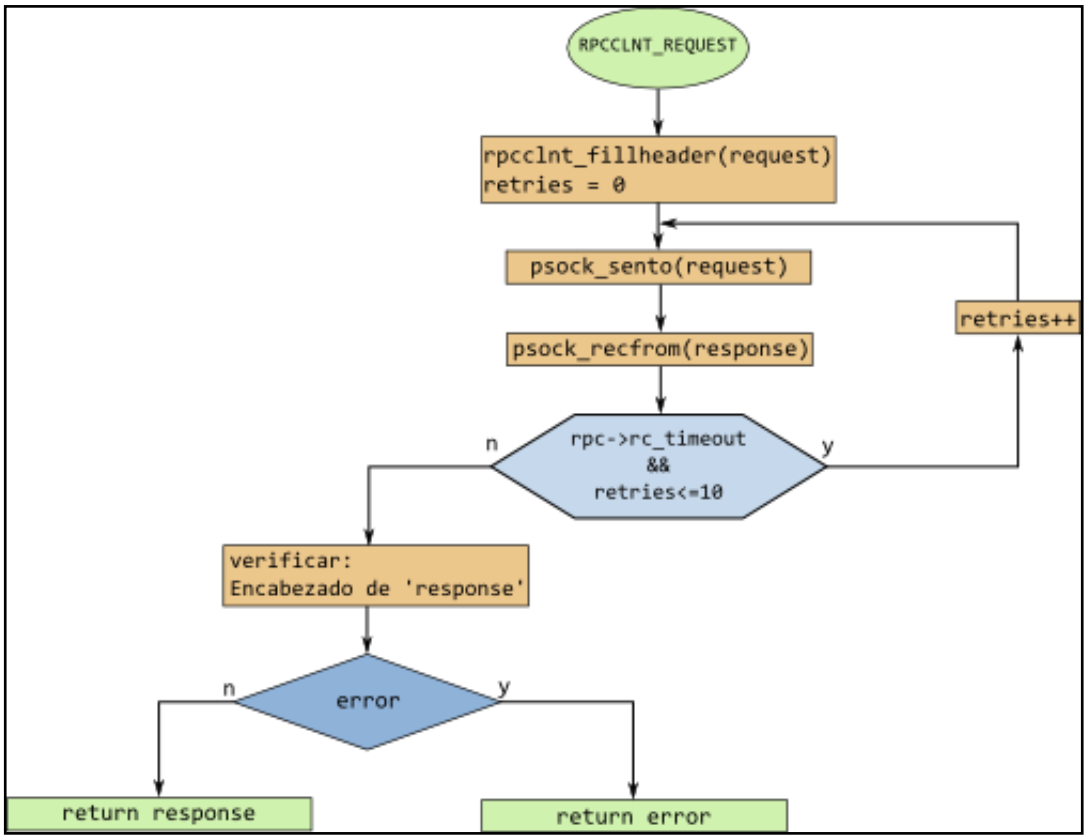


Figura 5.22 Diagrama de flujo de la operación RPCCLNT_REQUEST en NuttX RTOS.

En primer lugar, la función escribe en el encabezado del mensaje RPC de llamada: el programa, el procedimiento y la versión específicos que se está

solicitando. Luego, el programa, el procedimiento y la versión específicos que se está solicitando. Luego entra en un ciclo donde se envía el mensaje de llamada con la función *psock_sendto()* y se recibe en el mensaje de respuesta con la función *psock_recvfrom()*. En caso de existir algún error en alguna de estas dos funciones, el ciclo vuelve a empezar hasta alcanzar un número definido de intentos (NFS_RETRANS). Si no existe ningún error de esta índole, se procede a verificar el encabezado del mensaje de respuesta RPC verificando que no haya ocurrido un error dentro el proceso solicitado. De haber un error, se retorna el error al usuario y de no haberlo, se entrega el mensaje de respuesta a la función madre que lo solicito en la variable *response*.

5.2.4.2 RPCCLNT_CONNECT

Esta función se encarga de inicializar el socket para una conexión RPC y de realizar el montaje con el servidor.

La función de RPCCLNT_CONNECT recibe un parámetro de entrada:

- *FAR struct rpcclnt *rpc*: Estructura que contiene el estado de RPC en NuttX RTOS.

La salida de esta operación retorna un valor de cero si se realizó correctamente o retorna un valor negativo indicando el error ocurrido. La figura muestra el diagrama de flujo de la operación RPCCLNT_ CONNECT:

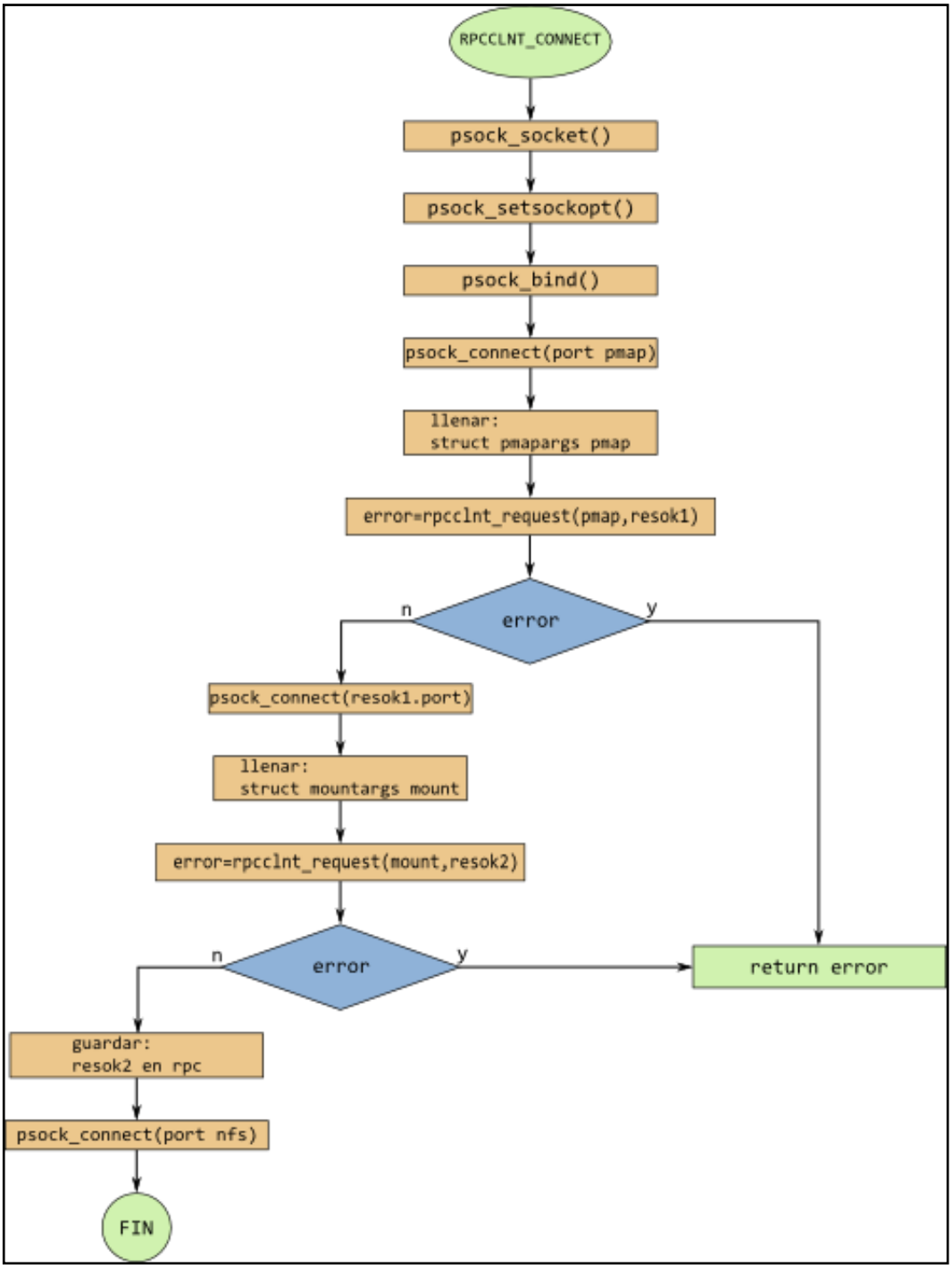


Figura 5.23 Diagrama de flujo de la operación RPCCLNT_ CONNECT en NuttX RTOS.

Inicialmente, esta función crea el socket con la función *psock_socket(),* luego le agrega opciones usando la función *psock_setsockopt() y* le asigna un numero de puerto local con *psock_bind()*. Seguidamente, se usa *psock_connect()* para asignar el valor del puerto remoto igual al del servicio PMAP.

Posteriormente, envía una petición RPC que contiene la información del procedimiento de GETPORT del protocolo PMAP, para obtener el número de puerto del servicio de montaje (MOUNTD). Se vuelve a utilizar *psock_connect()* para cambiar el número de puerto al del servicio MOUNTD y se envía una petición RPC con información del procedimiento de MOUNT del protocolo MOUNT, el cual contiene la ruta del directorio que se pretende montar. La respuesta a esta petición se copia en la estructura *rpcclnt* de la instancia y finalmente se vuelve a utilizar el la *psock_connect()* para establecer el puerto al número de puerto que utiliza NFS.

5.2.4.3 RPCCLNT_UMOUNT

Esta función se encarga de realizar el desmontaje del servidor.

La función de RPCCLNT_UMOUNT recibe un parámetro de entrada:

- *FAR struct rpcclnt *rpc*: Estructura que contiene el estado de RPC en NuttX RTOS.

La salida esta operación retorna un valor de cero si se realizó correctamente o retorna un valor negativo indicando el error ocurrido. La figura muestra el diagrama de flujo de la operación RPCCLNT_UMOUNT:

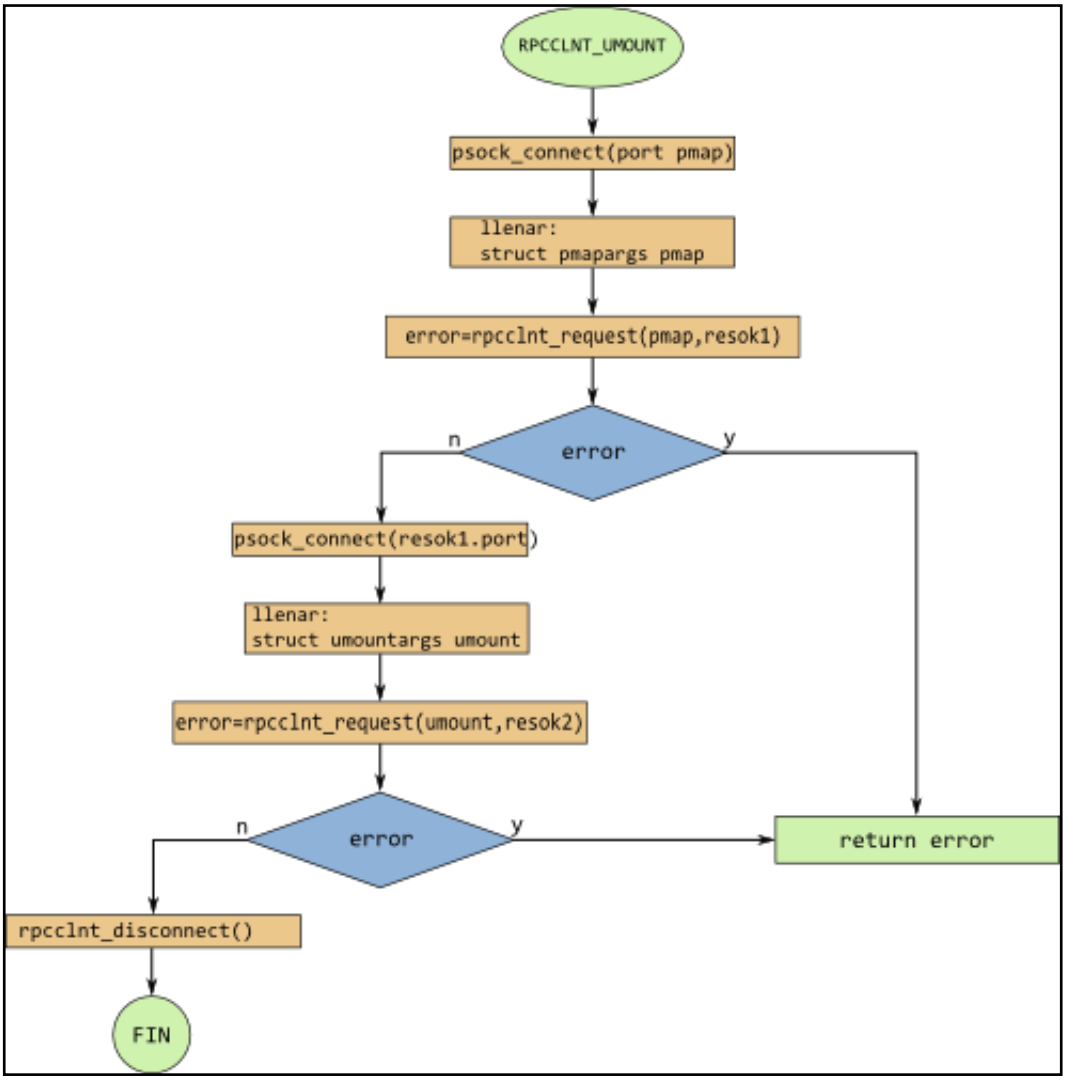


Figura 5.24 Diagrama de flujo de la operación RPCCLNT_UMOUNT en NuttX RTOS.

Similar a la a operación RPCCLNT_CONNECT, primero se usa *psock_connect()* para asignar el valor del puerto del servicio PMAP. Luego se envía una petición RPC que contiene la información del procedimiento de GETPORT del protocolo PMAP, de esta manera, se obtiene el número de puerto del servicio de montaje (MOUNTD). Nuevamente, se utiliza *psock_connect()* para cambiar el número de puerto al del servicio MOUNTD y se envía una petición RPC con la información del procedimiento de UMOUNT del protocolo MOUNT, el cual contiene la ruta del directorio que se pretende desmontar. Finalmente se desconecta el socket con la función *rpcclnt_connect().*

5.2.5 PSOCK

Este módulo contiene una serie de funciones que permiten el manejo de la entrega y de la recepción de paquetes de datos provenientes del periférico de red, mediante el mecanismo conocido como *"socket"*, el cual se define como un extremo dentro de una comunicación de red.

Además, brinda características de la capa transporte del modelo OSI, por lo que estas funciones se encargan de agregar el empaquetamiento correcto a los datos que serán intercambiados.

La nueva función de NFS se realizó utilizando el protocolo de transporte UDP, debido a que el cliente NFS se diseñó para trabajar dentro de una red de área local, la cual posee tasas de pérdidas muy bajas y presenta la característica de ser más rápido que TCP.

Las funciones del NuttX RTOS utilizadas en la solución son las siguientes:
- psock_socket(): Crea un socket para la comunicación.
- *psock_setsockopt()*: Establece las opciones del socket.
- *psock_bind()*: Le asigna al socket la dirección IP local y un numero de puerto.
- *psock_connect()*: Establece el socket del tipo UDP o TCP para realizar la debida conexión con el socket del servidor.
- *psock_close():* Realiza la operación de cierre en la instancia del socket.
- *psock_sendto()*: Utiliza la conexión del tipo UPD o TCP del socket para enviar los datos.
- *psock_recvfrom()*: Recibe los datos del socket servidor, utilizando el tipo de conexión (UDP o TCP) usada en el socket local.

5.2.6 Red y Hardware

En estos módulos, el NuttX RTOS se encarga trasladar la información con su respectiva encapsulación, así como la aplicación de los distintos protocolos de

enrutamiento y demás, de manera que siga la lógica del esquema del modelo OSI.

Cabe destacar que, en el último modulo (Hardware), el RTOS se encarga de aplicar las características específicas, configuradas con anterioridad en la construcción del programa, de la arquitectura que se está utilizando. En este caso, la arquitectura de la tarjeta TI/Luminary Stellaris LM3S6965 Evaluation Board [4] y la EAGLE 100 Single Board [6].

Capítulo 6 Análisis de Resultados

Este capítulo presenta los resultados obtenidos de las diferentes pruebas desarrolladas con el cliente NFS implementado en las dos plataformas. Además, a partir de los resultados, se les realiza un análisis detallado. Es importante aclarar que los resultados obtenidos son iguales en las dos tarjetas, por lo que solo se documenta el resultado de una plataforma.

Cabe destacar, que para monitorear las pruebas se utilizó el software Wireshark. El programa Wireshark es un analizador de protocolos open-source que tiene como principal objetivo el análisis del tráfico de red. Este posee una interfaz sencilla e intuitiva que facilita el desglose por capas de cada uno de los paquetes capturados.

6.1 Comandos "*nfsmount*" y "*umount*"

Para verificar el correcto funcionamiento del comando de montaje "*nfsmount"*, inicialmente, se debe configurar el servidor a utilizar como se explica en el apéndice A.2.

Durante la elaboración de las pruebas, se colocaron archivos y directorios dentro del directorio compartido de manera que se puede observar un funcionamiento correcto. El directorio compartido se encuentra en la trayectoria */expor/shared*.

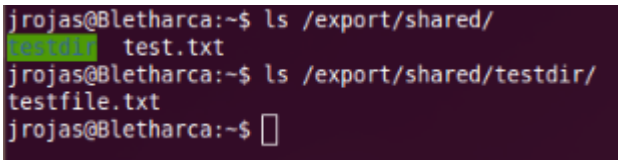


Figura 6.1 Directorio /expor/shared con archivos en el servidor con Ubuntu 10.04.

Inicialmente se puede observar en la carpeta de montaje */mnt* del NUTTX RTOS que no existe ningún directorio montado, como se observa a continuación:

Figura 6.2 Directorio /mnt del NuttX RTOS con ningún montaje.

Se procede a realizar la ejecución del comando "*nfsmount*" en NSH. Siguiendo el formato del comando como se muestra en el apéndice A.3, se nombra el punto de montaje *nfs,* la ruta corresponde a */mnt/nfs*. Posteriormente, se puede observar en la Figura 6.3 que la carpeta *nfs* ha sido creada y que se encuentra todo el contenido de la carpeta compartida */export/shared* del servidor.

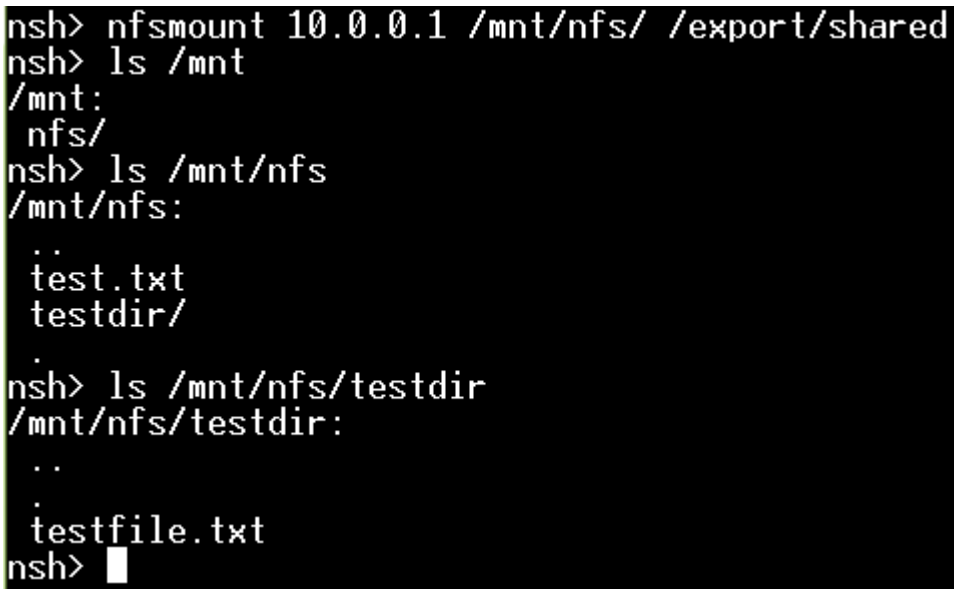
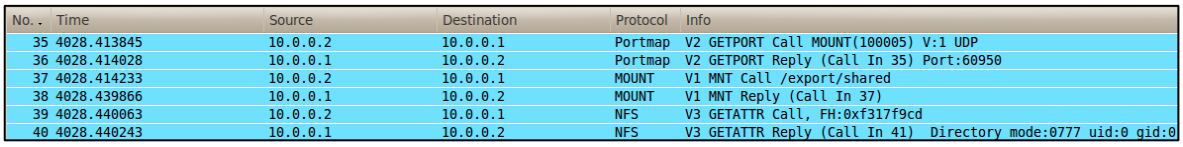


Figura 6.3 Ejecución del comando "*nfsmount*" en el NSH.

Como se puede observar, la ejecución de la operación de montaje fue completada a un 100%, sin errores de ninguna índole.

Utilizando la herramienta Wireshark, se pueden observar los mensajes RPC enviados y recibidos en el servidor durante el proceso de montaje, tal como se muestra a continuación:



No. .	Time	Source	Destination	Protocol	Info
35	4028.413845	10.0.0.2	10.0.0.1	Portmap	V2 GETPORT Call MOUNT(100005) V:1 UDP
36	4028.414028	10.0.0.1	10.0.0.2	Portmap	V2 GETPORT Reply (Call In 35) Port:60950
37	4028.414233	10.0.0.2	10.0.0.1	MOUNT	V1 MNT Call /export/shared
38	4028.439866	10.0.0.1	10.0.0.2	MOUNT	V1 MNT Reply (Call In 37)
39	4028.440063	10.0.0.2	10.0.0.1	NFS	V3 GETATTR Call, FH:0xf317f9cd
40	4028.440243	10.0.0.1	10.0.0.2	NFS	V3 GETATTR Reply (Call In 41) Directory mode:0777 uid:0 gid:0

Figura 6.4 Mensajes RPC durante la ejecución del comando "*nfsmount*".

En la Figura 6.4, se puede observar que se ha ejecutado en forma correcta la rutina RPCCLNT_CONNECT descrita en la sección 5.3.4.3, la cual es la encargada de enviar las peticiones RPC al servicio PMAP y MOUNTD. Así como también la petición RPC del procedimiento GETATTR, demuestra que la operación NFS_BIND fue ejecutada correctamente.

La ejecución de la operación de montaje funciona de la manera esperada, lo que indica que esta fue completada a un 100% de forma satisfactoria.

Por otra parte, es importante destacar, que el comando "*nfsmount*" cuenta con limitaciones. Debido a que el cliente NFS está enfocado hacia un sistema embebido, se optó por eliminar las opciones extra como modificar el tiempo de retrasmisión, el típico de protocolo de transporte y otras que se muestra en [24]. Esto para reducir el tamaño del código, así como el uso de memoria en ejecución. Por otro lado, muchos de los parámetros se establecieron por defecto como lo estipula el protocolo debido a que no existen en NuttX RTOS.

Igualmente, si se ejecuta el comando de "umount" se puede observar que se desmonta el punto de montaje *nfs,* dejando la carpeta de montajes nuevamente vacía.

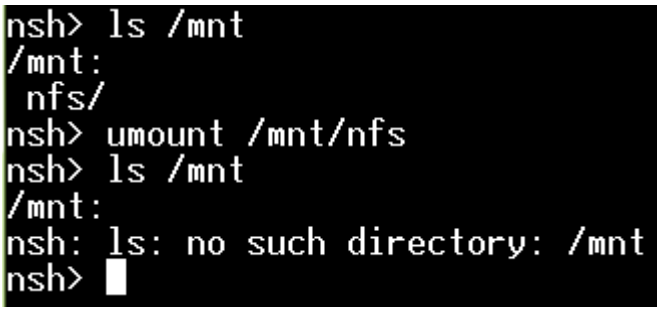
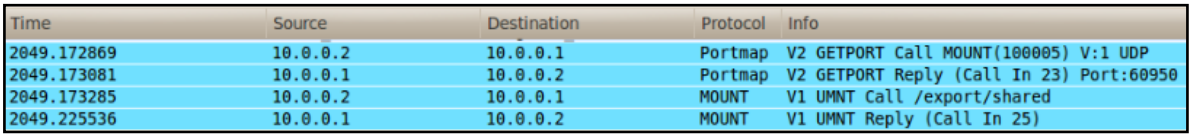


Figura 6.5 Ejecución del comando "*umount*" en el NSH.

En la Figura 6.6, se observa el envío de las peticiones RPC a los servicios PMAP y MOUNTD, de esta forma, se verifica que la operación NFS_UNBIND funciona correctamente.



Time	Source	Destination	Protocol	Info
2049.172869	10.0.0.2	10.0.0.1	Portmap	V2 GETPORT Call MOUNT(100005) V:1 UDP
2049.173081	10.0.0.1	10.0.0.2	Portmap	V2 GETPORT Reply (Call In 23) Port:60950
2049.173285	10.0.0.2	10.0.0.1	MOUNT	V1 UMNT Call /export/shared
2049.225536	10.0.0.1	10.0.0.2	MOUNT	V1 UMNT Reply (Call In 25)

Figura 6.6 Mensajes RPC durante la ejecución del comando "*umount*".

Se realizó una prueba donde se ejecutaban las operaciones de "*nfsmount*" y "*umount*" 10 veces consecutivas en las dos plataformas. Los resultados se muestran en la tabla 6.1.

Tabla 6.1 Ejecuciones correctas de los comandos *"nfsmount*" y "*umount*" para cantidad 10 intentos consecutivos.

	Comando *nfsmount*	**Comando *umount***	**Porcentaje (%)**
Cantidad correcta	**10**	**10**	**100**
Cantidad errónea	**0**	**0**	**100**

Según estos resultados, se puede determinar que estos comandos funcionan correctamente. Sin embargo, esta prueba no es válida para el "*nfsmount*". Al habilitar el modo debug en el NuttX RTOS, se observa que al ejecutar el comando *nfsmount* al inicializar las plataformas, ocurre un error 11 (Intentar de nuevo).

Como en el cliente NFS posee una rutina de realizar una cantidad de reintentos del montaje, este error no es crítico ya que no afecta posteriormente. Este error se da debido a que inicialmente el servidor no conoce las direcciones IP de las plataformas, por lo tanto, la rutina envía un mensaje ARP para determinar los clientes y agregarlos en la lista de enrutamientos. El mensaje ARP se muestra en la Figura 6.8.

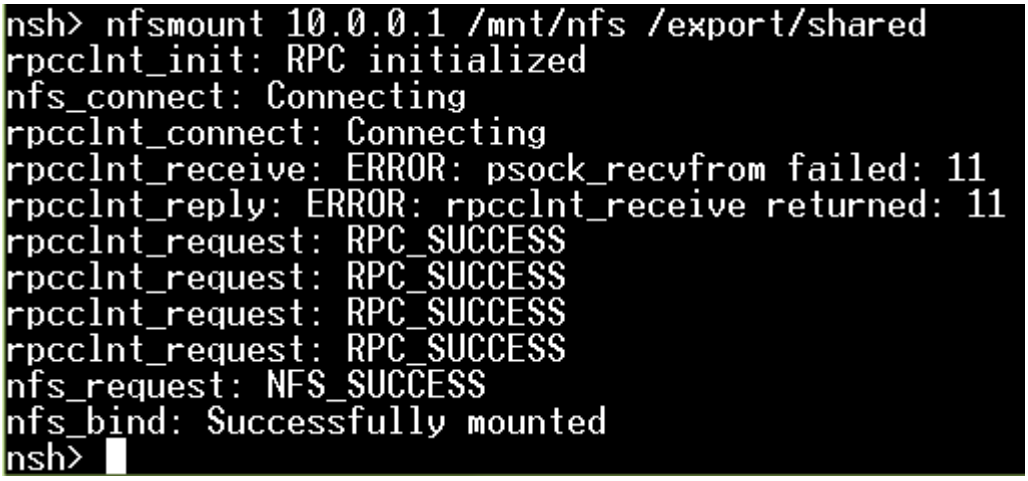
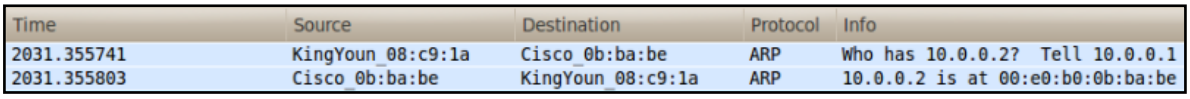


Figura 6.7 Error en la primera ejecución del comando *"nfsmount"*



Time	Source	Destination	Protocol	Info
2031.355741	KingYoun_08:c9:1a	Cisco_0b:ba:be	ARP	Who has 10.0.0.2? Tell 10.0.0.1
2031.355803	Cisco_0b:ba:be	KingYoun_08:c9:1a	ARP	10.0.0.2 is at 00:e0:b0:0b:ba:be

Figura 6.8 Mensaje ARP enviado por el servidor.

6.2 Creación, modificación y eliminación de un archivo

A pesar de las diferentes operaciones que permite realizar el cliente NFS en NuttX RTOS, se realizó una prueba que permitiera comprobar el funcionamiento básico del manejo del sistema de archivos una vez que este fuera montado. Por tanto, se optó por realizar la creación, modificación y eliminación de un mismo archivo.

6.2.1 Creación del archivo

Inicialmente se crea un archivo *prueba.txt*. Así como se muestra a continuación:

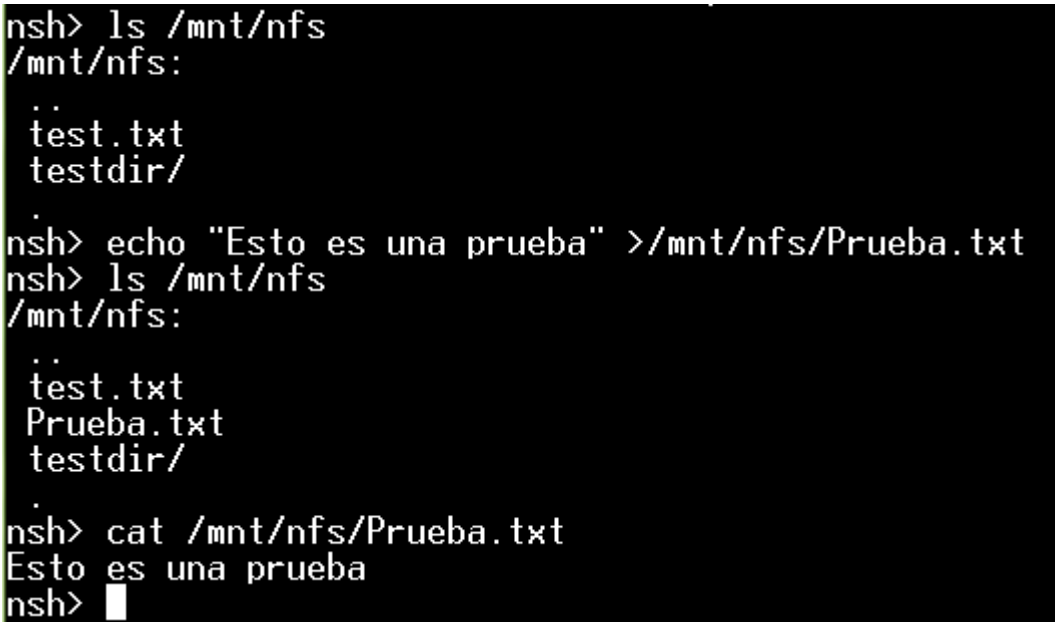


Figura 6.9 Creación del archivo prueba.txt con contenido en el NSH.

Como se observa, el comando "*echo*" en el NSH, es el comando realiza la llamada a la operación NFS_OPEN, la cual está encargada de realizar la creación del archivo si este no existe. Este comando "*echo*" permite crear un archivo, adjuntando a la misma vez el contenido de este.

Al ingresar a la carpeta compartida en el servidor, se observa que el archivo fue creado con éxito.

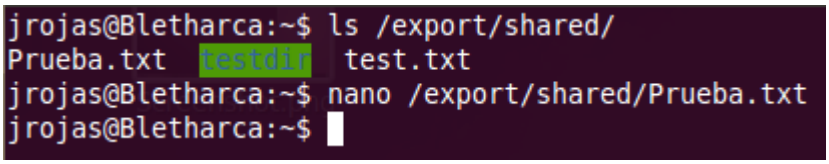


Figura 6.10 Archivo *prueba.txt* en el servidor NFS.

En las operaciones realizadas en la figura 6.9, se utiliza de manera implícita las operaciones NFS_WRITE, NFS_READDIR y NFS_READ. La escritura del archivo utiliza la rutina NFS_WRITE, mientras que la lectura del archivo mediante el comando *ls* utiliza la operación NFS_READDIR, finalmente, el comando "*cat*", el cual permite concatenar todo los caracteres del archivo y desplegarlos en el NSH, utiliza la operación NFS_READ.

Los mensajes NFS visualizados en Wireshark, se observa a continuación:

Time	Source	Destination	Protocol	Info
5208.154218	10.0.0.2	10.0.0.1	NFS	V3 LOOKUP Call, DH:0x33f6392c/Prueba.txt
5208.154306	10.0.0.1	10.0.0.2	NFS	V3 LOOKUP Reply (Call In 49) Error:NFS3ERR_NOENT
5208.154612	10.0.0.2	10.0.0.1	NFS	V3 CREATE Call, DH:0x33f6392c/Prueba.txt Mode:GUARDED
5208.201778	10.0.0.1	10.0.0.2	NFS	V3 CREATE Reply (Call In 51)
5208.202264	10.0.0.2	10.0.0.1	NFS	V3 WRITE Call, FH:0x12daf862 Offset:0 Len:20 FILE_SYNC
5208.243861	10.0.0.1	10.0.0.2	NFS	V3 WRITE Reply (Call In 53) Len:20 FILE_SYNC
5215.230764	10.0.0.2	10.0.0.1	NFS	V3 READDIR Call, FH:0x33f6392c
5215.230883	10.0.0.1	10.0.0.2	NFS	V3 READDIR Reply (Call In 55) .. test.txt Prueba.txt testdir .
5255.204528	10.0.0.2	10.0.0.1	NFS	V3 LOOKUP Call, DH:0x33f6392c/Prueba.txt
5255.204621	10.0.0.1	10.0.0.2	NFS	V3 LOOKUP Reply (Call In 79), FH:0x12daf862
5255.204925	10.0.0.2	10.0.0.1	NFS	V3 READ Call, FH:0x12daf862 Offset:0 Len:20
5255.205007	10.0.0.1	10.0.0.2	NFS	V3 READ Reply (Call In 81) Len:20

Figura 6.11 Mensajes NFS durante la creación y verificación del archivo *Puebas.txt*

Para observar la estructura de los mensajes del procedimiento CREATE se realiza un énfasis en los mensajes de llamada y respuesta. Se realiza un enfoque principalmente en los encapsulados de los protocolos implementados (RPC y NFS). Esta estructura se muestra a continuación:

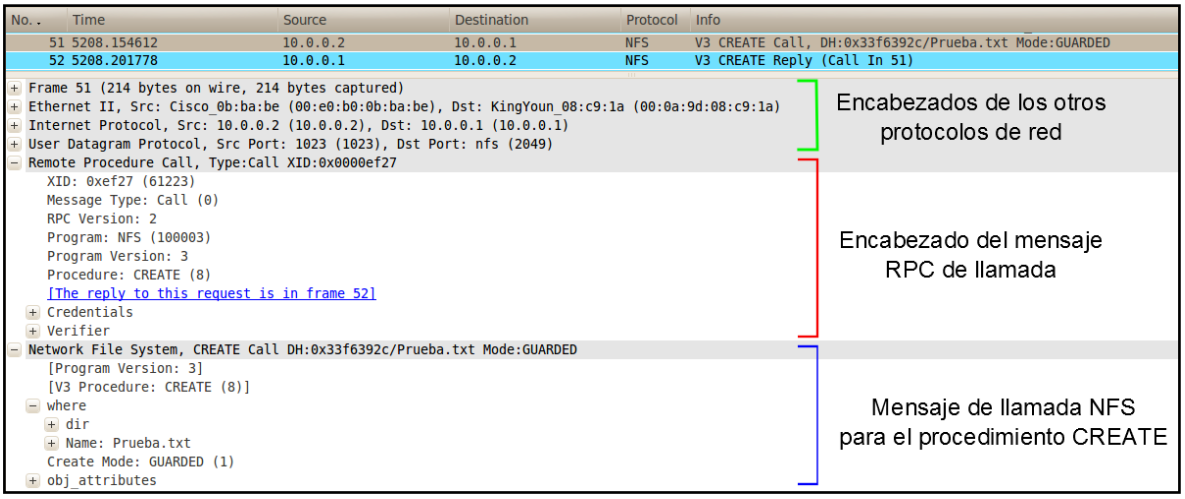


Figura 6.12 Formato del mensaje de llamada del procedimiento CREATE del protocolo NFS.

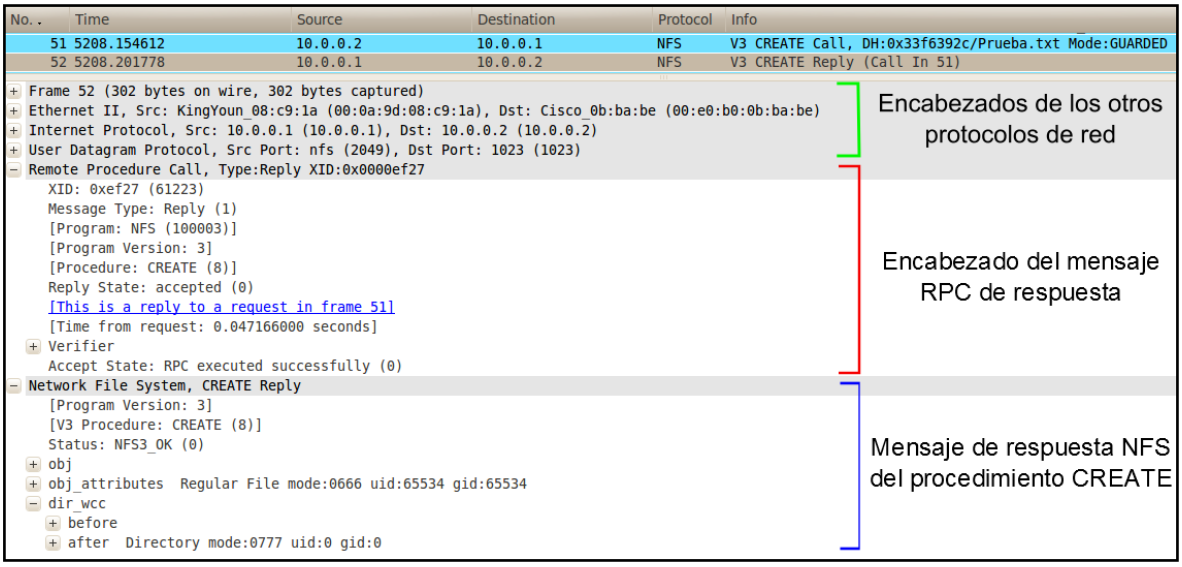


Figura 6.13 Estructura del mensaje de respuesta al procedimiento CREATE del protocolo NFS.

Se puede observar en las dos figuras anteriores, la manera en que los mensajes han sido particionados en los distintos encapsulados de las capas del modelo OSI. Entre los distintos encapsulados, se pueden observar el encapsulado para el protocolo Ethernet, el protocolo IPv4, el protocolo de UDP y los dos encapsulados que se implementaron en el proyecto, los protocolos RPC y NFS.

Según la Figura 6.12, el encabezado del mensaje RPC de llamada, envía los parámetros necesarios para que el servidor ejecute el procedimiento CREATE del protocolo NFS. Además de los parámetros, este encabezado envía el nombre de los archivos y otros respectivos atributos específicos de este procedimiento.

En la figura 6.13 se observa la estructura del mensaje de respuesta.

Las estructuras específicas de los protocolos RPC y NFS, como se comentó en una sección anterior, están especificadas en el RFC 1831 [22] y en el RFC 1813[20], respectivamente.

6.2.2 Modificación del archivo

En la ejecución de esta operación se utilizó la rutina NFS_RENAME. Esta operación se realiza con el comando "*mv*" en el NSH. Se utilizó el archivo

Prueba.txt y se le cambió el nombre a *Cambio.txt.* Los resultados de la prueba se muestran en la siguiente figura.

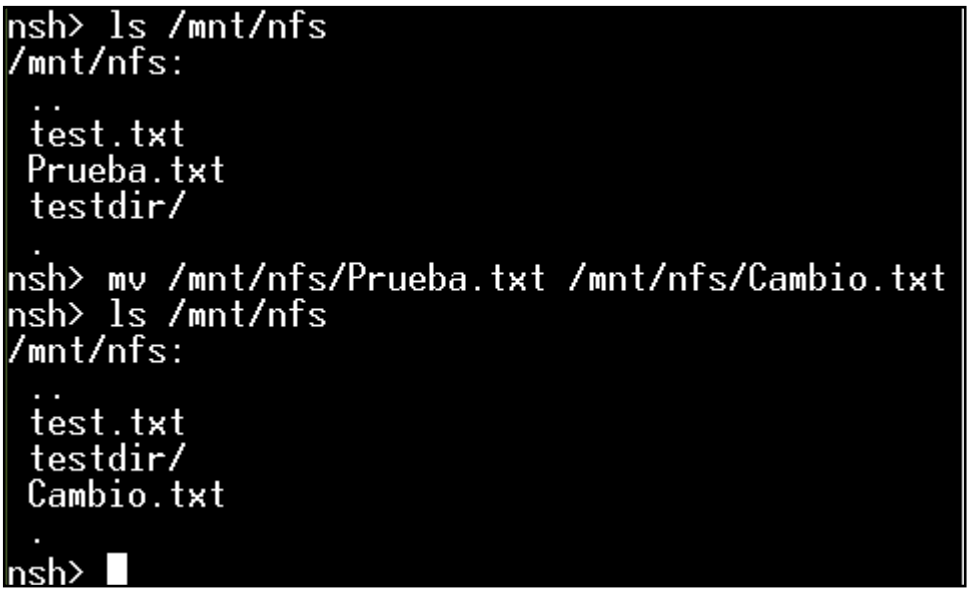


Figura 6.14 Renombramiento del archivo *prueba.txt* a *Cambio.txt* en el NSH.

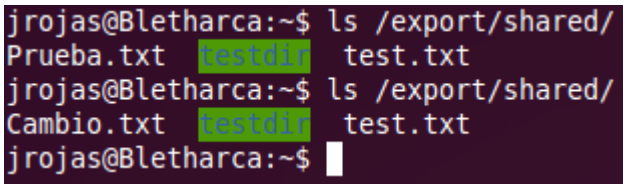


Figura 6.15 Cambio del archivo prueba.txt a Cambio.txt visto en el servidor.

A continuación se muestra la estructura de los encabezados RPC y los datos NFS del procedimiento RENAME.

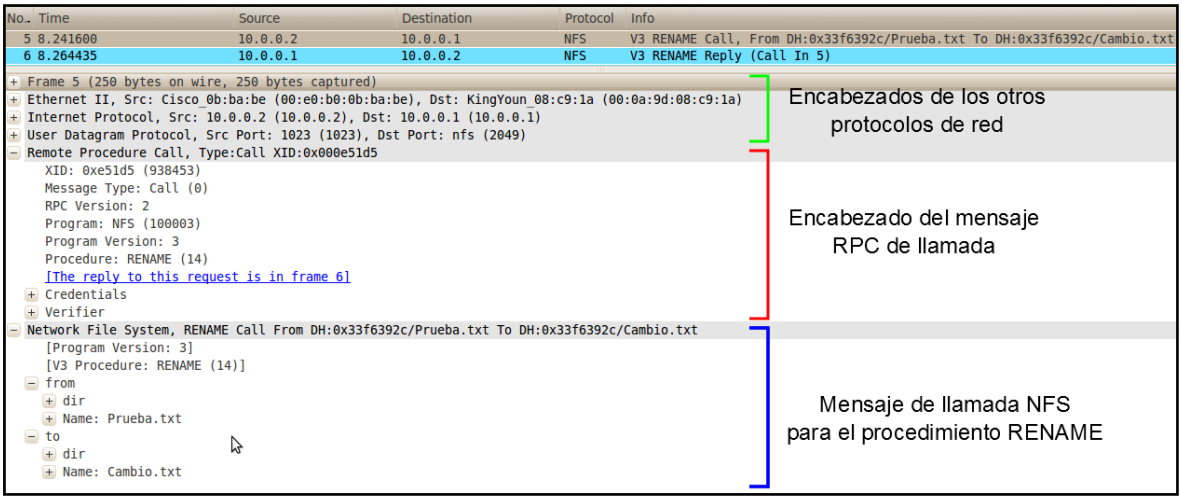


Figura 6.16 Formato del mensaje de llamada del procedimiento RENAME del protocolo NFS.

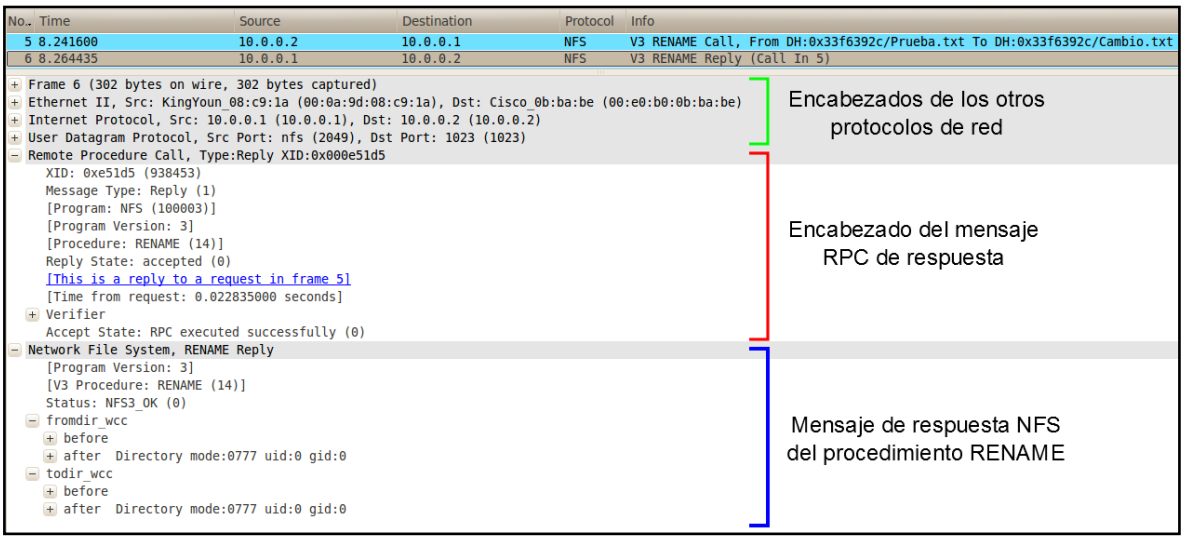


Figura 6.17 Estructura del mensaje de respuesta al procedimiento RENAME del protocolo NFS.

6.2.3 Eliminación del archivo

Para la verificación del correcto funcionamiento de la rutina NFS_REMOVE se ejecutó el comando "*rm*" en el NSH. En esta prueba se eliminó el archivo *Cambio.txt,* creado en la sección 6.2.2. A continuación, se observa los resultados de esta prueba:

Figura 6.18 Eliminación de archivo Cambio.txt en el NSH.

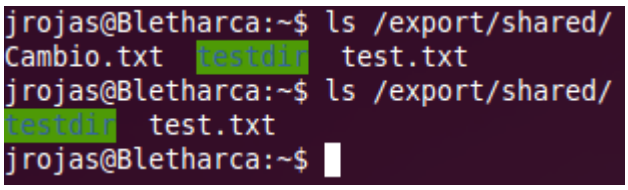


Figura 6.19 Archivo Cambio.txt eliminado en el servidor.

En las siguientes figuras, se observa los mensajes NFS de llamada y respuesta así como la estructura de los encabezados RPC y los datos NFS del procedimiento REMOVE.

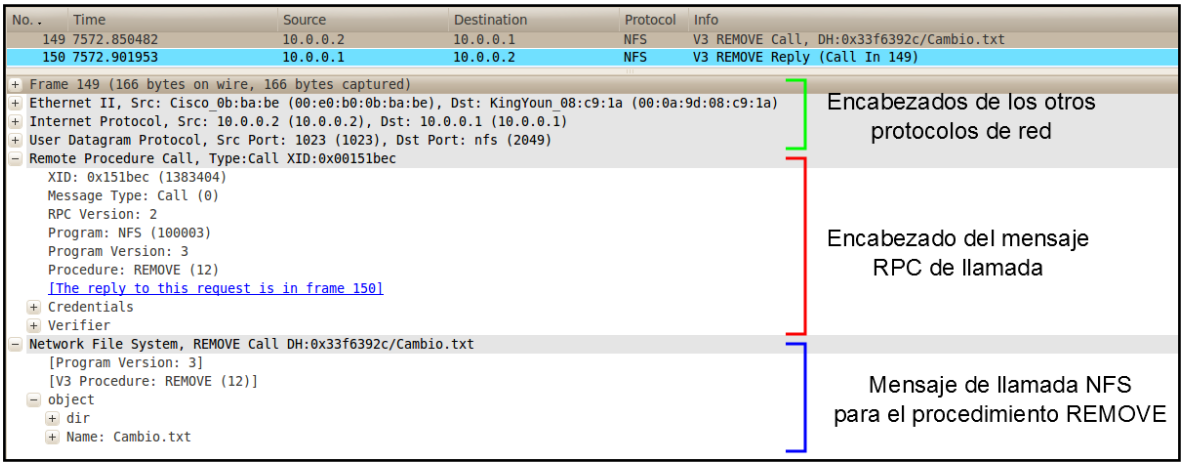


Figura 6.20 Formato del mensaje de llamada del procedimiento REMOVE del protocolo NFS.

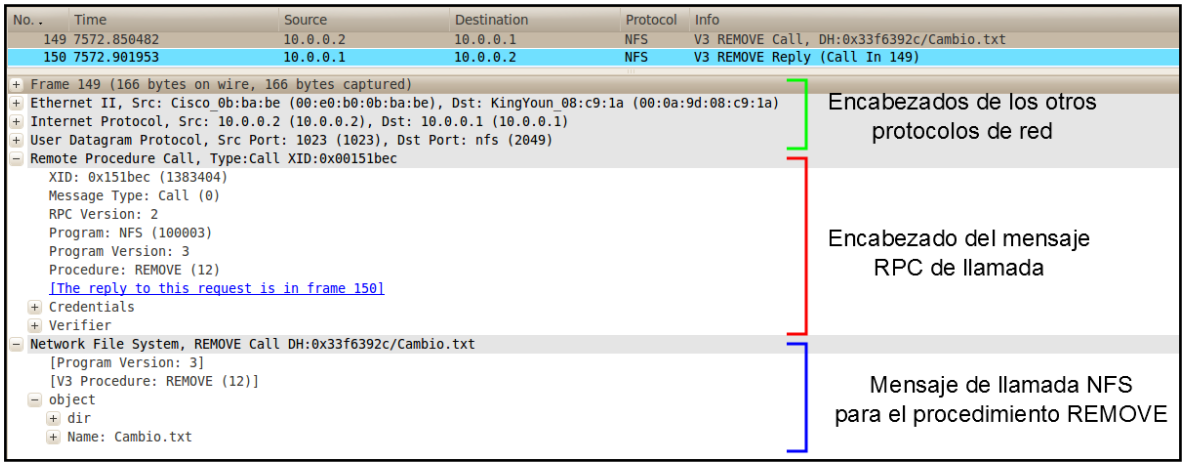


Figura 6.21 Estructura del mensaje de respuesta al procedimiento REMOVE del protocolo NFS.

En la siguiente tabla se muestra las estadísticas del funcionamiento de las operaciones en las dos tarjetas.

Tabla 6.2 Funcionamiento de las operaciones implementadas sobre las dos plataformas utilizadas.

Operaciones	Stellaris LM3S6965	Eagle 100
NFS_OPEN	✔	✔
NFS_READ	✔	✔
NFS_WRITE	✔	✔
NFS_OPENDIR	✔	✔
NFS_READDIR	✔	✔
NFS_BIND	✔	✔
NFS_UNBIND	✔	✔
NFS_STATFS	✔	✔
NFS_REMOVE	✔	✔
NFS_MKDIR	✔	✔
NFS_RMDIR	✔	✔
NFS_RENAME	✔	✔
Total	**12**	**12**
Porcentaje (%)	**100**	**100**

Como se puede observar, la función de cliente NFS funciona correctamente en las dos plataformas, por lo tanto esta función en compatible con las plataformas que soportan el NuttX RTOS.

Finalmente, según lo discutido en esta sección, se alcanzó los primeros dos objetivos específicos del proyecto.

6.3 Consumo de memoria

Se realizaron mediciones de la memoria utilizada por el NuttX RTOS, de esta manera, se obtiene el valor del consumo máximo de la función NFS, y a su vez, verifica el correcto desmontaje del sistema.

A continuación se muestra el estado de la memoria del sistema NuttX RTOS al iniciar:

Figura 6.22 Estado de la memoria al iniciar el NuttX RTOS.

Se observa que el sistema operativo al inicializar, utiliza 12368 bytes. Este valor es el valor de memoria usado sin haber ejecutado el comando "*nfsmount*"

Luego se procede a realizar el montaje NFS. Al verificar el estado de la memoria después del montaje, se obtiene los siguientes resultados:

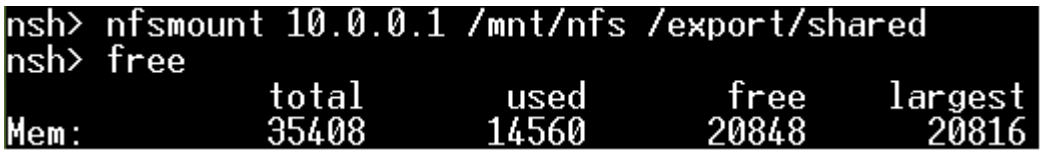


Figura 6.23 Estado de la memoria después de ejecutar el comando "*nfsmount*".

Posterior al comando de montaje, se realizaron una serie de operaciones de las implementadas en la solución y nuevamente se verificó el estado de la memoria. El valor de 14560 bytes corresponde al valor de memoria usada luego de realizar el comando de montaje. Finalmente, se realizó el desmontaje del punto de montaje. Donde los resultados se muestran a continuación.

Figura 6.24 Estado de la memoria después de ejecutar el comando “umount”

A continuación se adjunta una tabla resumen con el estado de memoria en las diferentes situaciones para las dos tarjetas utilizadas:

Tabla 6.3 Estado de memoria en NuttX RTOS**.**

Plataforma	**Memoria usada sin *nfsmount* (Bytes)**	**Memoria usada con *nfsmount* (Bytes)**	**Memoria usada por *nfsmount* (Bytes)**
TI/Luminary Stellaris LM3S6965	12368	14560	2192
EAGLE 100	12368	14560	2192

Según la tabla anterior, la nueva función de cliente NFS sólo consume 2192 bytes durante un montaje. Este valor corresponde al tamaño de la estructura *nfsmount*. El manejo de las otras estructuras y variables son dinámicas, por lo que no consumen memoria RAM. Por lo tanto, este diseño cumple con la expectativa de conservación de memoria para sistemas embebidos. Además, según los resultados de la Figura 6.24, después del desmontaje, el sistema operativo posee el mismo valor de memoria antes del montaje.

Con lo anteriormente discutido, se logró alcanzar el objetivo específico tres, demostrando una optimización de 56% con respecto al valor planteado de 5 kBytes.

6.4 Aplicaciones futuras

Inicialmente, el proyecto desarrollado no tenía una aplicación específica, sin embargo, después de la implementación, esta función podrá ser utilizada para aplicaciones como recopilación de datos, para la información de configuración para un MCU, para actualizaciones de software, para el control de una "granja" de MCU's, entre muchas otras.

Estas aplicaciones son solo algunas de las que se podrían implementar, por lo que las posibilidades de innovación están abiertas. Por ejemplo, en el caso de recopilación de datos, podríamos implementar el cliente NFS en el proyecto realizado por la Ing. Carmen Chan llamado "*Sistema Empotrado de recolección, fusión, procesamiento y envío de datos basado en GNU/Linux*" [25]. De manera tal, que en vez de usar la memoria de la plataforma para introducir la recolección los datos en un archivo para ser enviados a un servidor, se podría realizar una aplicación que tome los datos de la recolección y al estar disponibles, escriba el archivo necesario directamente sobre el servidor .

Otro posible ejemplo, sería si se utiliza el cliente NFS para dar información de configuración a un MCU, este podría tratarse de un robot (o cualquier sistema) que no se tenga disposición físicamente, pero se encuentre conectado a la red y posea un programador integrado.

Así no se tiene la necesidad de tomar el MCU, configurarlo y programarlo en una computadora aparte para luego agregárselo de nuevo al robot.

Capítulo 7 Conclusiones y recomendaciones

7.1 Conclusiones

- El NuttX RTOS cuenta con una nueva función de cliente NFS que le permite acceder archivos remotos como si estos fueran locales.
- El cliente NFS implementado en NuttX RTOS permite realizar al menos las operaciones básicas del protocolo de red NFS.
- El diseño de la nueva función de cliente NFS cuenta con una implementación enfocada al uso óptimo de la memoria.
- El sistema de archivos de red en el cliente NFS utiliza siempre un espacio constante de memoria igual 2192 bytes.
- El cliente NFS permite incrementar la memoria de la plataforma soportada por NuttX RTOS, en forma virtual, al tamaño de memoria disponible en el servidor.
- Todas las acciones ejecutadas por el cliente NFS, se ven reflejadas en forma instantánea en el servidor y viceversa.

7.1 Recomendaciones

- Implementar y probar el diseño final en todas las demás plataformas soportadas por NuttX RTOS.
- Agregar al cliente NFS diseñado, la opción de utilizar el protocolo de transporte TCP, permitiendo así una optimización de rendimiento, en términos de red, al diseño.

- Agregar y actualizar las bibliotecas del NuttX RTOS, para poder soportar los demás procedimientos del protocolo NFS que no se lograron implementar, debido a esta limitación.

- Implementar un servidor NFS para el NuttX RTOS, de manera que la plataforma pueda ser un pequeño servidor embebido.

Bibliografía

[1] Differencebetween.net. "*Difference Between NFS and CIFS".* 2010. Disponible en: http://www.differencebetween.net/technology/difference-between-nfs-and-cifs/

[2] NX-Engineering*. ''Specialists in NuttX Embedded Systems"*. Disponible en: http://www.nx-engineering.com/index.html

[3] NX-Engineering. ''*NuttX RTOS".* 2011. Disponible en: http://nuttx.sourceforge.net/

[4] Texas Instrument. "*Stellaris® LM3S6965 Evaluation Board: User's manual".* 2012. Disponible en: http://www.ti.com/lit/ug/spmu029a/spmu029a.pdf

[5] The FreeBSD Foundation**. "***Manual de FreeBSD: Capítulo 29. Networking avanzado"*. Disponible en: http://www.freebsd.org/doc/es/books/handbook/network-nfs.html.

[6] Micromint. "*EAGLE 100 Single Board Computer: User's manual".* 2012. Disponible en: http://www.micromint.com/docs/Eagle-100-Users-Manual.pdf

[7] The Internet Engineering Task Force**. "***NFS Version 3 Protocol Specification"*. Disponible en: http://www.ietf.org/rfc/rfc1813.txt

[8] The FreeBSD Foundation**. "***FreeBSD: The power to serve"*. Disponible en: http://www.freebsd.org/es/

[9] Proyecto OpendBSD**. "***OpenBSD4.8: Free, functional and secure"*. Disponible en: http://www.openbsd.org/es/

[10] BSD Organization**. "***BSD"*. Disponible en: http://www.bsd.org/

[11] NX-Engineering. ‘‘*NuttShell User Guide”.* 2012. Disponible en: http://nuttx.sourceforge.net/NuttShell.html

[12] Sun Microsystems**. “***RFC 1094 - NFS: Network File System Protocol specification”*. 1989. Disponible en: http://www.faqs.org/rfcs/rfc1094.html

[13] O'Reilly & Associates, Inc**. “***Managing NFS and NIS”*. Disponible en: http://docstore.mik.ua/orelly/networking_2ndEd/nfs/

[14] Caldera Internacional, Inc. “*NFS resources”*. Disponible en: http://osr507doc.sco.com/en/PERFORM/NFS_rsc.html

[15] Brent Baccala. “*Connected: An Internet Encyclopedia*”. 1997. Disponible en: http://www.freesoft.org/CIE/Topics/115.htm

[16] Coulouris G., Dollimore J. & Kindberg T. “*Distributed Systems: Concepts and Design”*. 1994. Disponible en: http://www.cdk3.net/rmi/Ed2/SunRPC.pdf

[17] Javvin Company. ‘‘*RPC: Remote Procedure Call protocol”*. Disponible en: http://www.javvin.com/protocolRPC.html

[18] Cisco redes. ‘‘*Modelo OSI y TCP/IP”*. Disponible en: http://www.ciscoredes.com/tutoriales/60-modelo-osi-y-tpc-ip.html

[19] Ordenadores-y-portatiles.com. ‘‘*¿Cómo funciona el modelo OSI?”.* Disponible en: http://www.ordenadores-y-portatiles.com/modelo-osi.html

[20] Sun Microsystems, Inc. ‘‘*NFS Version 3 Protocol Specification”.* 1995. Disponible en: http://www.faqs.org/rfcs/rfc1813.html

[21] ARM. "*Cortex-M3 Processor".* Disponible en: http://www.arm.com/products/processors/cortex-m/cortex-m3.php

[22] Sun Microsystems, Inc. "*RPC: Remote Procedure Call Protocol Specification Version 2".* 1995. Disponible en: http://www.faqs.org/rfcs/rfc1831.html

[23] Wireshark foundation. "*Wireshark". 2012.* Disponible en: http://www.wireshark.org/

[24] Red Hat. "*Common NFS Mount Options".* Disponible en: http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Storage_Administration_Guide/s1-nfs-client-config-options.html

[25] Carmen Chan. Sistema Empotrado de recolección, fusión, procesamiento y envío de datos basado en GNU/Linux. Tesis de licenciatura,
Escuela de Ingeniería Electrónica, ITCR, Cartago, Costa Rica, Junio 2012.

Apéndices

A.1. Glosario y abreviaturas

- NSH: Nutt Shell
- RTOS : Real-Time Operating System
- BSD: Berkeley Software Distribution
- NFS : Network File System
- RPC: Remote Procedure Call
- VFS: Virtual File System
- MCU : Microcontrolador
- IP : Internet Protocol
- FAT : File Allocation Table
- ROM: read-only filesystem
- RAM: Random-access memory
- OSI : Open System Interconnection
- ARP: Address Resolution Protocol
- TCP: Transmision Control Protocol
- UDP: User Datagram Protocolo
- APDU: Unidad de datos en la capa de aplicación.
- PPDU: Unidad de datos en la capa de presentación.
- SPDU: Unidad de datos en la capa de sesión.
- TPDU (segmento): Unidad de datos en la capa de transporte.
- Paquete o Datagrama: Unidad de datos en el nivel de red.
- Trama: Unidad de datos en la capa de enlace.
- Bits: Unidad de datos en la capa física.

A.2. Configuring the NFS server (Ubuntu)

Setting up the server will be done in two steps: First, setting up the configuration file for NFS, and then starting the NFS services. But first, you need to install the nfs server on Ubuntu with these two commands:

```
# sudo apt-get install nfs-common
```

```
# sudo apt-get install nfs-kernel-server
```

After that, we need to make or choose the directory we want to export from the NFS server. In our case, we are going to make a new directory called /export.

```
# sudo mkdir /export
```

It is important that /export directory allow access to everyone (777 permissions) as we will be accessing the NFS share from the client with no authentication.

```
# sudo chmod 777 /export
```

When all this is done, we will need to edit the configuration file to set up an NFS server: `/etc/exports`. This file contains a list of entries; each entry indicates a volume that is shared and how it is shared. For more information for a complete description of all the setup options for this file you can check in the man pages (`man export`).

An entry in /etc/exports will typically look like this:

```
directory machine1(option11,option12)
```

So for our example we export /export to the client 10.0.0.2 add the entry:

```
/export 10.0.0.2(rw)
```

In our case we are using all the default options except for the ro that we replaced with rw so that our client will have read and write access to the directory that we are exporting.

After we do all the require configurations, we are ready to start the server with the next command:

```
# sudo /etc/init.d/nfs-kernel-server start
```

Note: If you later decide to add more NFS exports to the /etc/exports file, you will need to either restart NFS daemon or run command exportfs.

```
# sudo /etc/init.d/nfs-kernel-server start
```

Or

```
# exportfs -ra
```

Now we can check if the export directory and our mount point is properly set up.

```
# sudo showmount -e
# sudo showmount -a
```

And also we can verify if NFS is running in the system with:

```
# rpcinfo –p
program vers proto   port
   100000   2   tcp    111  portmapper
   100000   2   udp    111  portmapper
   100011   1   udp   749  rquotad
   100011   2   udp   749  rquotad
   100005   1   udp    759  mountd
   100005   1   tcp    761  mountd
   100005   2   udp    764  mountd
   100005   2   tcp    766  mountd
   100005   3   udp    769  mountd
   100005   3   tcp    771  mountd
   100003   2   udp   2049  nfs
   100003   3   udp   2049  nfs
   300019   1   tcp    830  amd
   300019   1   udp    831  amd
   100024   1   udp    944  status
   100024   1   tcp    946  status
   100021   1   udp   1042  nlockmgr
   100021   3   udp   1042  nlockmgr
   100021   4   udp   1042  nlockmgr
   100021   1   tcp   1629  nlockmgr
   100021   3   tcp   1629  nlockmgr
   100021   4   tcp   1629  nlockmgr
```

Now your NFS sever is sharing `/export` directory to be accessed.

A.3. NFS Mount Command

The NuttShell (NSH) also supports a command called `nfsmount` that can be used to mount a remote file system via the NSH command line.

Command Syntax:

```
nfsmount <server-address> <mount-point> <remote-path>
```

Synopsis. The `nfsmount` command mounts a network file system in the NuttX psuedo filesystem. The `nfsmount` will use NFSv3 UDP protocol to mount the remote file system.

Command Line Arguments. The `nfsmount` takes three arguments:

1. The `<server-address>` is the IP address of the server exporting the file system you wish to mount. This implementation of NFS for the NuttX RTOS is only for a local area network, so the server and client must be in the same network.
2. The `<mount-point>` is the location in the NuttX pseudo filesystem where the mounted volume will appear. This mount point can only reside in the NuttX pseudo filesystem. By convention, this mount point is a subdirectory under `/mnt`. The mount command will create whatever psuedo directories that may be needed to complete the full path (but the full path must not already exist).
3. The `<remote-path>` is the file system `/` directory being exported from server. This `/` directory must have been configured for exportation on the server before when the NFS server was set up.

After the volume has been mounted in the NuttX pseudo filesystem, it may be access in the same way as other objects in the file system.

Example. Suppose the the NFS server has been configured to export the directory `/export/shared`. The the following command would mount that file system (assuming that the target also has privileges to mount the file system).

```
NuttShell (NSH)
nsh> ls /mnt
/mnt:
nsh: ls: no such directory: /mnt
nsh> nfsmount 10.0.0.1 /mnt/nfs /export/shared
nsh> ls -l /mnt/nfs
/mnt/nfs:
 drwxrwxrwx   4096 ..
 drwxrwxrwx   4096 testdir/
 -rw-rw-rw-      6 ctest.txt
 -rw-r--r--     15 btest.txt
 drwxrwxrwx   4096 .
nsh> echo "This is a test" >/mnt/nfs/testdir/testfile.txt
nsh> ls -l /mnt/nfs/testdir
/mnt/nfs/testdir:
 -rw-rw-rw-     21 another.txt
 drwxrwxrwx   4096 ..
 drwxrwxrwx   4096 .
 -rw-rw-rw-     16 testfile.txt
nsh> cat /mnt/nfs/testdir/testfile.txt
This is a test
```