NuttX ESP32 Lab01

AndoniXXR

21 de noviembre de 2025

Contents

1 Introducción **1**

2 Arquitectura del Sistema **3**
- 2.1 Descripción del Proyecto . . . 3
- 2.2 Características Implementadas . . . 4

3 Configuración del Sistema y Entorno **5**
- 3.1 Configuración de NuttX . . . 5
- 3.2 Configuración del Host (PC Gateway) . . . 5
- 3.3 Guía de Uso . . . 6

4 Solución de Problemas **7**

5 Documentación de la API (Código Fuente) **9**
- 5.1 Aplicación PC (lab01_pc.c) . . . 9
- 5.2 Aplicación ESP32 (lab01_main.c) . . . 11

CHAPTER 1

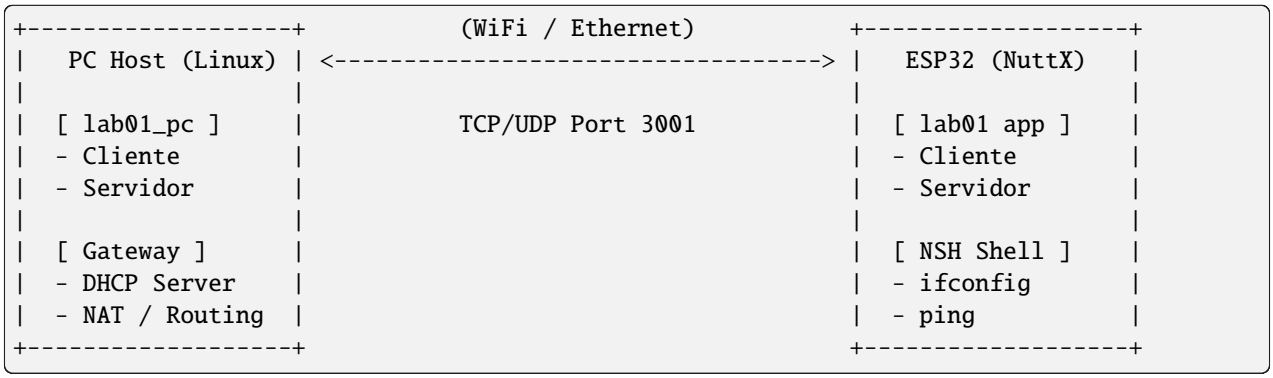
Introducción

Esta es la documentación técnica generada automáticamente para el proyecto de portabilidad de Laboratorio 01 a NuttX en ESP32. El proyecto consiste en una aplicación cliente/servidor TCP/UDP que se ejecuta en el microcontrolador ESP32 y se comunica con una contraparte en PC.

CHAPTER 2

Arquitectura del Sistema

El sistema se compone de dos nodos principales que se comunican a través de una red TCP/IP.



- **PC Host**: Actúa como la estación de desarrollo y también como Gateway de red. Ejecuta la versión Linux de la aplicación (*lab01_pc*) para pruebas de integración.
- **ESP32**: Ejecuta el RTOS NuttX. La aplicación *lab01* está integrada en el sistema operativo y se lanza desde la consola NSH.

2.1 Descripción del Proyecto

El objetivo de este laboratorio es demostrar la capacidad del sistema operativo NuttX ejecutándose en un SoC ESP32 para manejar comunicaciones de red estándar (TCP/IP).

Se ha implementado una aplicación llamada `lab01` que puede funcionar en dos modos: * **Modo Servidor**: Escucha peticiones de cálculo matemático simple (suma, resta, multiplicación, división, módulo) y devuelve el resultado. * **Modo Cliente**: Permite al usuario ingresar operaciones matemáticas y enviarlas a un servidor remoto.

2.2 Características Implementadas

1. **Soporte Dual TCP/UDP**: La aplicación puede configurarse en tiempo de ejecución para usar cualquiera de los dos protocolos de transporte.
2. **Calculadora Remota**: El protocolo de aplicación es texto plano simple. El cliente envía una operación (ej. `5+3`) y el servidor responde con el resultado (ej. `8`).
3. **Comando EXIT**: Se implementó un mecanismo de cierre limpio. Si el cliente envía la cadena `EXIT`, el servidor cierra la conexión actual (en TCP) o termina su ejecución (si así se desea), y el cliente termina su proceso.
4. **Logging**: Ambas partes (Cliente y Servidor) registran en consola los mensajes enviados y recibidos con marcas de tiempo.

CHAPTER 3

Configuración del Sistema y Entorno

3.1 Configuración de NuttX

Para lograr la funcionalidad requerida, se realizaron modificaciones específicas en la configuración del kernel de NuttX (*defconfig*) y en el sistema de construcción.

Características Habilitadas:

- **Networking (TCP/IP)**: Se habilitó la pila de red completa (IPv4), soporte para sockets TCP y UDP, y controladores para la interfaz Ethernet/Wi-Fi del ESP32.
- **NuttShell (NSH)**: Se habilitó la consola interactiva del sistema, permitiendo la ejecución de comandos y scripts.
- **Comandos del Sistema**: Se incluyeron utilidades esenciales para la gestión y diagnóstico: * `ifconfig`: Gestión de interfaces de red (configuración de IP, máscara, gateway). * `ping`: Diagnóstico de conectividad ICMP. * `ls`, `ps`, `free`: Gestión de archivos y procesos. * `reboot`: Reinicio del sistema. * `date`: Gestión de fecha y hora del sistema.
- **Aplicaciones de Usuario**: Se registró la aplicación `lab01` en el sistema de construcción (*apps/examples/lab01*), haciéndola accesible directamente desde la línea de comandos de NSH.

3.2 Configuración del Host (PC Gateway)

Para probar la conectividad en un entorno controlado, el PC se configura como un Gateway/Router, proporcionando servicios de red al ESP32.

Herramientas Utilizadas:

- **Python 3**: Para ejecutar el script de configuración y el servidor DHCP simple.
- **iptables**: Para configurar NAT (Network Address Translation) y permitir que el ESP32 acceda a redes externas a través del PC.
- **iproute2 (ip)**: Para la configuración de direcciones IP y enlaces en Linux.

Scripts de Configuración:

El repositorio incluye scripts para automatizar esta tarea:

1. **setup_gateway.py**: Script principal en Python. * Configura la IP estática en la interfaz LAN del PC (ej. `192.168.50.1`). * Habilita el reenvío de paquetes (IP Forwarding) en el kernel de Linux. * Configura reglas de `iptables` para enmascarar el tráfico (NAT) saliente por la interfaz WAN. * Ejecuta un servidor DHCP básico para asignar automáticamente una IP al ESP32 (ej. `192.168.50.2`).

   Uso: `sudo python3 setup_gateway.py <INTERFAZ_WAN> <INTERFAZ_LAN>`

2. **setup_network.sh**: Alternativa en Bash para configuración manual de red y NAT.

3.3 Guía de Uso

3.3.1 Comandos en ESP32 (NuttShell)

Una vez que el sistema NuttX ha arrancado y tienes acceso a la consola NSH a través del puerto serie:

1. **Verificar Red**: Asegúrate de tener IP.

```
nsh> ifconfig
```

2. **Ejecutar como Servidor**:

```
# Iniciar servidor TCP en puerto 3001
nsh> lab01 server tcp 3001
```

3. **Ejecutar como Cliente**:

```
# Conectar a un servidor en PC (ej. 192.168.1.50)
nsh> lab01 client tcp 3001 192.168.1.50
```

3.3.2 Comandos en PC (Linux)

Primero compila la aplicación de contraparte:

```
gcc lab01_pc.c -o lab01_pc
```

1. **Ejecutar como Servidor** (para probar el Cliente ESP32):

```
./lab01_pc server tcp 3001
```

2. **Ejecutar como Cliente** (para probar el Servidor ESP32):

```
./lab01_pc client tcp 3001 <IP_DEL_ESP32>
```

CHAPTER 4

Solución de Problemas

1. El ESP32 no obtiene dirección IP * Verifique que el servidor DHCP en el PC esté corriendo (*ps aux | grep python*). * Asegúrese de que el cable Ethernet esté conectado o la red Wi-Fi configurada correctamente. * Intente asignar una IP estática manualmente: *ifconfig eth0 192.168.50.2*.

2. No hay conexión entre PC y ESP32 (Ping falla) * Verifique el firewall del PC (*sudo iptables -L*). Asegúrese de que se permitan conexiones entrantes en el puerto 3001. * Compruebe que ambos dispositivos están en la misma subred.

3. Error «Connection refused» * Asegúrese de que el servidor (ya sea en PC o ESP32) esté ejecutándose *antes* de iniciar el cliente. * Verifique que el puerto 3001 no esté siendo usado por otra aplicación.

CHAPTER 5

Documentación de la API (Código Fuente)

5.1 Aplicación PC (lab01_pc.c)

Este archivo contiene la lógica para el cliente/servidor que se ejecuta en el host (Linux). Incluye funciones para manejo de sockets BSD estándar y parsing de argumentos.

Aplicación de contraparte para PC (Linux) del Laboratorio 01.

Este programa actúa como Cliente o Servidor TCP/UDP para probar la conectividad y funcionalidad de la aplicación corriendo en el ESP32 con NuttX.

Author
AndoniXXR

Date
2025

Defines

BUFFER_SIZE

Tamaño del buffer para envío y recepción de mensajes.

Functions

static void **get_timestamp**(char *buffer, size_t size)

Obtiene la fecha y hora actual formateada.

Parámetros

- **buffer** – Buffer donde se escribirá la cadena de tiempo.
- **size** – Tamaño del buffer.

static void **log_msg**(const char *direction, const char *host, const char *socket_type, const char *protocol, const char *description)

Imprime un mensaje de log con marca de tiempo en la consola.

Parámetros

- **direction** – Dirección del mensaje («>» enviado, «<» recibido).
- **host** – Host remoto o local.
- **socket_type** – Tipo de socket («client» o «server»).
- **protocol** – Protocolo usado («TCP» o «UDP»).
- **description** – Contenido del mensaje o descripción del evento.

static int **parse_args**(int argc, char *argv[], struct *pc_args_s* *args)

Parsea los argumentos de la línea de comandos.

Parámetros

- **argc** – Número de argumentos.
- **argv** – Array de argumentos.
- **args** – Puntero a la estructura donde se guardarán los resultados.

Devuelve

0 si tiene éxito, -1 si faltan argumentos obligatorios.

static void **calculate**(const char *input, char *output)

Realiza una operación matemática básica basada en una cadena de entrada.

Soporta operaciones: +, -, *, /, %.

Parámetros

- **input** – Cadena de entrada (ej. «5+3»).
- **output** – Buffer donde se escribirá el resultado o mensaje de error.

static int **run_client**(struct *pc_args_s* *args)

Ejecuta la lógica del cliente.

Se conecta al servidor (TCP) o prepara el socket (UDP), lee de stdin y envía los comandos.

Parámetros

args – Argumentos de configuración.

Devuelve

0 en éxito, 1 en error.

static int **run_server**(struct *pc_args_s* *args)

Ejecuta la lógica del servidor.

Escucha en el puerto especificado y procesa las peticiones de cálculo.

Parámetros

args – Argumentos de configuración.

Devuelve

0 en éxito, 1 en error.

int **main**(int argc, char *argv[])

Punto de entrada principal de la aplicación PC.

Parámetros

- **argc** – Número de argumentos.
- **argv** – Array de argumentos.

Devuelve

0 en éxito, 1 en error.

struct **pc_args_s**

Estructura para almacenar los argumentos de línea de comandos.

Public Members

char ***protocol**

Protocolo a utilizar: «TCP» o «UDP».

char ***server_ip**

Dirección IP del servidor al que conectarse (modo cliente).

int **port**

Puerto de conexión o escucha.

char ***mode**

Modo de operación: «client» o «server».

5.2 Aplicación ESP32 (lab01_main.c)

Este archivo contiene la lógica principal de la aplicación NuttX. Se registra como una aplicación del sistema en NuttX y se invoca desde NSH. Utiliza la API de sockets compatible con POSIX de NuttX.

Aplicación principal para NuttX (ESP32) del Laboratorio 01.

Este programa implementa un cliente/servidor TCP/UDP que corre sobre NuttX. Permite realizar operaciones matemáticas simples enviadas desde un cliente o procesarlas si actúa como servidor.

Author

AndoniXXR

Date

2025

Defines

BUFFER_SIZE

Tamaño del buffer para envío y recepción de mensajes.

Functions

static void **get_timestamp**(char *buffer, size_t size)

Obtiene la fecha y hora actual formateada.

Parámetros

- **buffer** – Buffer donde se escribirá la cadena de tiempo.
- **size** – Tamaño del buffer.

static void **log_msg**(const char *direction, const char *host, const char *socket_type, const char *protocol, const char *description)

Imprime un mensaje de log con marca de tiempo en la consola (NSH).

Parámetros

- **direction** – Dirección del mensaje («>» enviado, «<» recibido).
- **host** – Host remoto o local.
- **socket_type** – Tipo de socket («client» o «server»).
- **protocol** – Protocolo usado («TCP» o «UDP»).
- **description** – Contenido del mensaje o descripción del evento.

static int **parse_args**(int argc, char *argv[], struct *args_s* *args)

Parsea los argumentos de la línea de comandos de NuttX.

Parámetros

- **argc** – Número de argumentos.
- **argv** – Array de argumentos.
- **args** – Puntero a la estructura donde se guardarán los resultados.

Devuelve

0 si tiene éxito, -1 si faltan argumentos obligatorios.

static void **calculate**(const char *input, char *output)

Realiza una operación matemática básica.

Parámetros

- **input** – Cadena de entrada (ej. «5+3»).
- **output** – Buffer donde se escribirá el resultado.

static int **run_client**(struct *args_s* *args)

Ejecuta la lógica del cliente en NuttX.

Parámetros

args – Argumentos de configuración.

Devuelve

0 en éxito, 1 en error.

static int **run_server**(struct *args_s* *args)

Ejecuta la lógica del servidor en NuttX.

Parámetros

args – Argumentos de configuración.

Devuelve
0 en éxito, 1 en error.

`int main (int argc, FAR char *argv[])`

Punto de entrada principal de la aplicación NuttX.

Parámetros

- **argc** – Número de argumentos.
- **argv** – Array de argumentos.

Devuelve
0 en éxito, 1 en error.

struct **args_s**

Estructura para almacenar los argumentos de línea de comandos en NuttX.

Public Members

char ***protocol**

Protocolo a utilizar: «TCP» o «UDP».

char ***server_ip**

Dirección IP del servidor al que conectarse (modo cliente).

int **port**

Puerto de conexión o escucha.

char ***mode**

Modo de operación: «client» o «server».