# Zenoh-Pico\_SylixOS移植报告

目录

[Zenoh-Pico\_SylixOS移植报告 1](#_Toc186731141)

[第一章 项目来源与目标 2](#_Toc186731142)

[1.1 项目来源 2](#_Toc186731143)

[1.2 移植目标 2](#_Toc186731144)

[1.3 风险评估 3](#_Toc186731145)

[第二章 背景介绍 5](#_Toc186731146)

[2.1 SylixOS简介 5](#_Toc186731147)

[2.2 Zenoh-Pico简介 6](#_Toc186731148)

[2.3 移植方案设计 9](#_Toc186731149)

[2.4 Zenoh-pico模块剖析 11](#_Toc186731150)

[第三章 目标完成情况 12](#_Toc186731151)

[第四章 移植过程 15](#_Toc186731152)

[4.1 SylixOS交叉编译 15](#_Toc186731153)

[4.2 编译zenoh-pico 19](#_Toc186731154)

[1. 构建CMake 19](#_Toc186731155)

[2. 编译工程 22](#_Toc186731156)

[第五章 测试与验证 24](#_Toc186731157)

[5.1 功能测试 24](#_Toc186731158)

[5.2 验证示例 28](#_Toc186731159)

[第六章 移植总结 32](#_Toc186731160)

## 项目来源与目标

本章将介绍Zenoh-Pico 项目的移植目标与风险评估，通过分析技术、资源、测试、依赖和市场等风险，识别在移植过程中可能面临的挑战，为后续的实施和优化提供指导。

### 1.1 项目来源

Zenoh-pico 是一款高性能的分布式通信中间件，设计初衷是为了统一数据在网络边缘和云端之间的管理。它不仅支持传统的发布/订阅模式，还支持查询和存储等功能，适用于工业物联网 (IIoT)、机器人控制和实时数据分发等场景。SylixOS 是一款实时操作系统 (RTOS)，广泛应用于嵌入式设备中，具有高可靠性、低延迟和良好的实时性。将 Zenoh-pico 移植到 SylixOS 上可以充分发挥这两者的优势，为实时通信提供高效的解决方案。

结合 Zenoh-pico 的高性能分布式通信能力和 SylixOS 的实时操作系统特性，为低资源环境中的分布式设备提供稳定可靠的数据交互方案。这种整合不仅能增强 SylixOS 平台的生态适配性，也能推动 Zenoh-pico 在实时嵌入式系统中的应用，为 IIoT 和边缘计算领域带来技术与商业价值的双重提升。

GitHub代码仓库：[eclipse-zenoh/zenoh-pico: Eclipse zenoh for pico devices](https://github.com/eclipse-zenoh/zenoh-pico)

官方：[Zenoh - The Zero Overhead, Pub/Sub, Store, Query, and Compute Protocol.](https://zenoh.io/)

### 1.2 移植目标

#### 1.2.1 功能目标

成功将 Zenoh-pico 移植到 SylixOS 平台，确保其核心功能（如数据发布、订阅、查询和数据存储）在 SylixOS 上能够正常运行。

确保 Zenoh-pico 的所有示例程序在 SylixOS 上均可编译、运行并通过测试。

验证 Zenoh-pico 的测试用例在 SylixOS 平台上能够正常执行且测试通过，全面验证点对点通信、发布/订阅、存储查询等功能的正确性，确保核心功能的完整性和稳定性。

#### 1.2.2 性能目标

保证移植后的 Zenoh-pico 在 SylixOS 平台上的运行效率接近或达到在 Linux 平台上的性能水平。

测试网络通信的延迟、吞吐量及资源使用情况，确保在 SylixOS 上达到实时通信需求。

#### 1.2.3 稳定性目标

确保 Zenoh-pico 在 SylixOS 上长时间运行稳定，能够处理高并发的网络请求。

验证内存管理和资源释放机制，确保无内存泄漏或资源使用异常。

### 1.3 风险评估

#### 1.3.1 技术风险

在移植过程中，Zenoh-Pico 可能会依赖特定的操作系统 API，例如文件系统、网络栈或线程管理等。但是，SylixOS 可能没有提供这些 API 的完整实现，导致功能无法正常运行。此外，数据结构的差异也可能导致数据处理出现问题，则需要特别关注数据类型的兼容性。

Zenoh-Pico 设计用于高效的数据传输和处理，但在 SylixOS 上，实时性能可能受到影响。尤其是在网络延迟和带宽有限的情况下，可能会出现性能瓶颈。SylixOS 的线程调度机制可能与其他操作系统不同，这可能会影响 Zenoh-Pico 的并发性能，导致响应时间延长。此外，Zenoh-Pico 可能依赖于特定的网络协议，如 TCP 或 UDP，来实现数据的发布与订阅。然而，SylixOS 的网络栈可能不完全支持这些协议，或者实现方式有所不同，这可能使得 Zenoh-Pico 的网络功能无法正常工作，从而影响整体性能和稳定性。

#### 1.3.2 资源风险

Zenoh-Pico 的内存管理可能依赖于 Linux 的某些机制，而 SylixOS 的内存分配和回收机制可能存在差异，这可能引发内存泄漏、碎片化或分配失败的问题。为此，需要在 SylixOS 上重点测试内存管理功能，并借助内存检测工具排查潜在的内存泄漏问题。

Zenoh-Pico 的部分功能可能依赖特定平台特性，而 SylixOS 平台可能无法完全支持。这可能导致部分功能在 SylixOS 上不可用或表现异常。对此，需要优先移植和测试核心功能，非核心功能视情况适配或暂时删除，并与 SylixOS 开发团队沟通以寻求技术支持。

#### 1.3.3 测试风险

在移植过程中，现有的测试用例可能并不能覆盖所有在 SylixOS 上可能出现的场景。这可能导致移植后出现未被发现的潜在问题，因此需要为 SylixOS 特有的功能和行为设计新的测试用例，以确保软件的可靠性。

## 第二章 背景介绍

本章将对项目的两大主体SylixOS和Zenoh-Pico的背景知识展开介绍，并阐述对本项目中需要使用的技术的调研。

### 2.1 SylixOS简介

SylixOS 是一款国产实时操作系统，专为嵌入式系统和工业控制应用设计，旨在提供高性能、高可靠性和强实时性的操作系统解决方案。其名称来源于“Synchro-Linux Operation System”的缩写，体现了其融合 POSIX 接口与实时内核技术的设计理念，目标是打造一个兼具灵活性和实时性的嵌入式操作系统平台。SylixOS 广泛应用于航空航天、轨道交通、能源工业、通信等多个领域，为这些行业的复杂需求提供支持。

SylixOS 的核心特点在于其强大的实时性能和对 POSIX 接口的高度兼容。其实时内核采用优先级抢占式调度，能够实现毫秒级甚至微秒级的调度延迟，满足对实时性有严格要求的工业场景。同时，SylixOS 提供多达 95% 的 POSIX 接口支持，使得开发者可以轻松地将基于 Linux 的应用移植到 SylixOS 平台上。这一设计显著降低了嵌入式开发的学习成本和开发难度，并为用户提供了高度灵活的开发环境。其模块化内核设计允许开发者根据实际需求对系统进行裁剪，并通过动态加载机制灵活扩展功能，以适应资源受限的嵌入式设备。

SylixOS 支持多种主流处理器架构，包括 ARM、x86、PowerPC 和 RISC-V，能够广泛适配不同的硬件平台。它还配备了高效的网络栈，支持 IPv4、IPv6、TCP 和 UDP 等标准协议，同时兼容工业通信协议（如 CAN 和 Modbus），为设备间的数据交互提供了强有力的支持。此外，SylixOS 支持多种文件系统，包括 FAT、YAFFS 和 ext 系列，能够满足不同设备的存储需求。这些特性使 SylixOS 成为嵌入式和工业控制应用中一种高效且灵活的操作系统选择。

作为国产实时操作系统的代表之一，SylixOS 已在多个关键领域中成功应用，例如 PLC 控制器、飞控系统、轨道交通信号系统和智能电网等。在这些应用场景中，SylixOS 的高可靠性和实时性得到了充分验证。此外，其广泛的硬件支持和灵活的模块化设计也使其在工业互联网和物联网领域展现出广阔的应用潜力。然而，SylixOS 在社区生态和技术支持方面仍有进一步发展的空间，与国际主流实时操作系统（如 VxWorks 和 FreeRTOS）相比，其高级功能和配套工具链的完善性仍需提升。尽管如此，SylixOS 作为国产操作系统，凭借自主可控的特点，已成为国内工业嵌入式系统发展的重要推动力。

### 2.2 Zenoh-Pico简介

Zenoh 是一种新型分布式通信中间件，专为满足物联网（IoT）、边缘计算和工业控制等场景中的复杂数据交互需求而设计。它以融合数据存储、查询、发布/订阅和点对点通信为核心，打破了传统通信中间件在数据传输方式上的限制。Zenoh将高效的数据交换与灵活的操作接口结合在一起，为开发者提供了一个高性能、低延迟、跨网络的通信平台。

Zenoh-pico 是 Zenoh 项目的一部分，是一种高性能的信息分发协议，旨在支持物联网和边缘计算领域的实时数据传输。Zenoh-Pico 结合了多种通信模式，包括点对点、发布/订阅和请求/响应，使得开发者可以根据具体需求选择最合适的通信方式。这种灵活性使得它可以广泛应用于各种场景，从简单的传感器数据采集到复杂的实时监控系统。其还具有良好的网络适应性，能够在不稳定的网络环境中保持高效的数据传输，支持多种网络协议，从而确保设备间的无缝连接。

此外，Zenoh-Pico 具备强大的数据管理能力，支持数据的持久化和检索功能。这意味着用户可以在分布式环境中有效地管理数据流，确保重要信息不会因网络波动而丢失。其高效的数据处理能力使得 Zenoh-Pico 特别适合用于需要快速响应和实时分析的应用，如智能制造、智能交通和智能城市等。

#### 2.2.1 部署单元

Zenoh 提供三种主要的部署单元，分别是平等节点（peers）、客户端（clients）和路由器（routers）。平等节点是指在对等网络中，所有节点都具备相等的地位和能力，能够直接进行数据传输和交换。这种设计使得网络更加灵活，降低了中心化的依赖，提高了系统的鲁棒性。

客户端是指用于请求和接收数据的终端设备，它们通过路由器连接到其他平等节点，实现数据的传递。客户端的设计允许用户方便地访问和处理数据，适应不同的应用需求。路由器则在 Zenoh 网络中扮演着关键角色，它负责管理数据流的转发和路由。路由器能够连接多个客户端和其他平等节点，确保信息能够高效且准确地在网络中传递。

结合这些组件，Zenoh 的设计使得不同设备能够在一个分布式环境中进行灵活的通信。这种架构不仅能够支持大规模的网络部署，还能够在多种网络拓扑（如星形、网状等）中有效工作，从而满足现代物联网和边缘计算的需求。通过这些关键概念的整合，Zenoh 能够提供高效、可靠的信息交换机制，推动智能设备之间的无缝协作。图1是zenoh提供的拓扑结构。

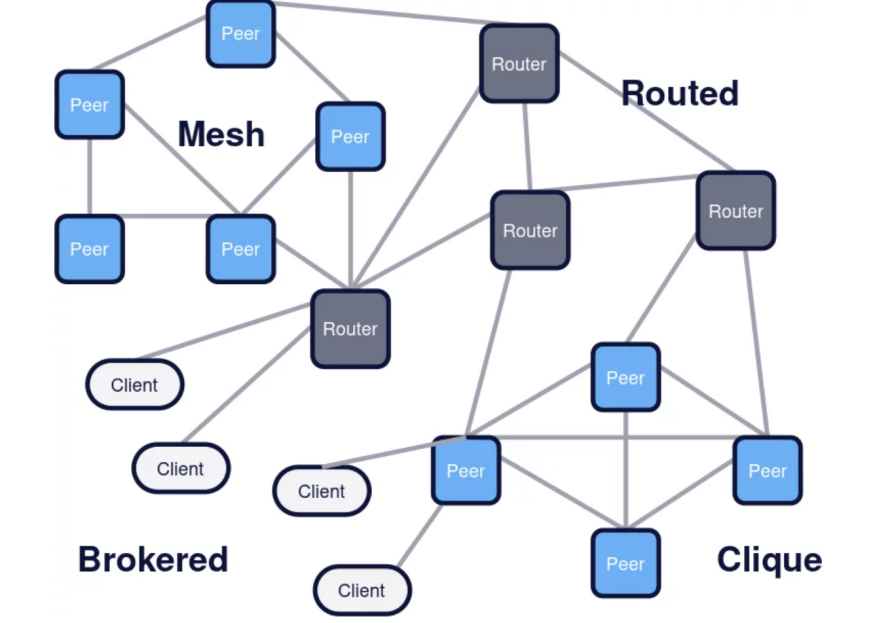


图1 zenoh提供的拓扑结构

#### 2.2.2 两级API

Zenoh 的架构层次及其核心组件如下图所示，它分为两大部分：Zenoh 和 Zenoh.net，并映射到常见的网络通信分层模型。zenoh提供两级 API，这种设计旨在简化用户与复杂网络系统的交互。

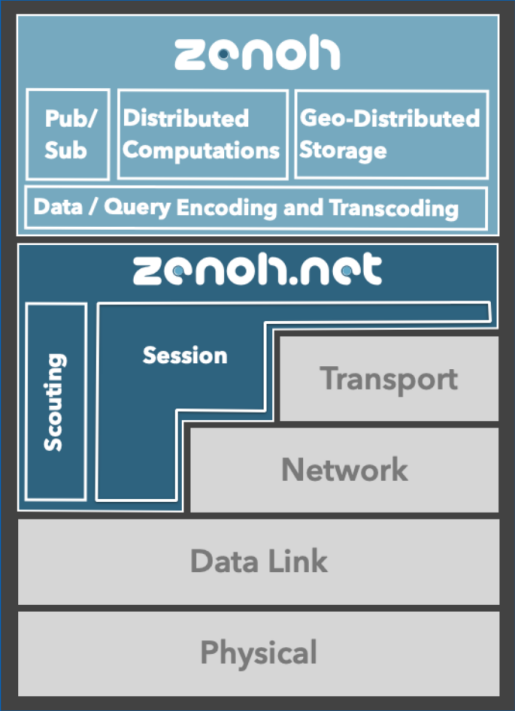


图2 zenoh架构层次

**（1）Zenoh（功能层）**

zenoh层为 Pub/Sub 和分布式查询提供了高级API，以更简单和更面向数据的方式提供与zenoh-net API相同的抽象，并提供所有构建块来创建分布式存储。zenoh层知道数据内容，并且可以应用基于内容的过滤和代码转换。

它处理数据表示编码和代码转换，并提供地理分布式存储和分布式计算值的实现。 zenoh原生支持一系列数据编码，例如JSON，Properties，Relational，Raw等，以及跨支持格式的转码。它还基于URI语法定义了规范的查询语法。zenoh层还提供了存储后端插件API，以简化第三方存储技术的集成。 当前支持的存储后端是Memory，MySQL，MariaDB，PostgreSQL，SQLite和InfluxDB。默认情况下，地理分布式存储在最终一致性下工作。 用户可以利用仲裁（Quorum）机制实现更强的一致性。

提供的原语：

**Put**：将实时数据推送到匹配的订阅者和存储。 （相当于zenoh.net write）

**Subscribe**：实时数据的订阅者。 （相当于zenoh.net订阅）

**Get**：从匹配的存储和演算中获取数据。 （相当于zenoh.net查询）

**Storage**：一个zenoh-net 订阅者侦听要存储的实时数据，以及一个zenoh.net 可查询以响应匹配的get请求的组合。

**eval**：能够响应请求的实体。通常用于按需提供数据或构建RPC系统。（相当于zenoh-net可查询）

**（2）Zenoh.net（网络层）**

Zenoh.net 是 Zenoh 的传输层基础，它与网络协议栈紧密集成，支持高效的数据通信。Zenoh.net 的设计高度模块化，可适配不同的网络协议和环境：

Zenoh.net支持点对点和路由通信，提供了允许Pub/Sub（Push）通信以及 Query/Reply (Pull) 通信的关键原语。它支持分段和有序可靠交付，并为 Discovery( ROS2 Discovery Service )提供了可插入的侦查抽象。zenoh-net定义并建立在会话协议的基础上，该会话协议为有序尽力而为（ best effort）和具有不受限制的 MTU 的可靠信道提供了抽象，而 MTU 与底层无关。Zenoh.net层仅关心数据传输，不关心数据内容或存储数据。

提供的原语：

**Write**：将实时数据推送给相对应的订阅者。

**Subscribe**：订阅实时数据。

**Query**：从匹配的查询对象中查询数据。

**Queryable**：能够响应查询的实体。

这种两级 API 结构使得开发者能够更灵活地选择适合自己应用的功能模块，同时也简化了复杂网络环境下的开发流程。通过这种设计，Zenoh 能够为用户提供强大的功能和灵活性，满足各种实时数据处理和存储需求，进一步推动物联网和边缘计算的发展。

#### 2.2.3 zenoh特性

**更好的协议支持**。在利用网络特定功能的同时将 zenoh 移植到不同类型的网络非常简单。用户提供的附件与数据和查询可以一起携带。 zenoh应用程序可以使用这些附件来扩展协议，或例如增加用户级别的安全性。

**通用的点对点和客户通信**。应用程序可以在运行时决定其行为类似于 对等方（peer） 还是 客户（client）。对等方在彼此之间路由信息，也可以代表客户进行路由，换句话说，对等方的行为类似于路由器。对等网络通信支持任意连接图，并在特殊情况下支持集团。

**基于Closure的发现**。侦察（scouting）协议支持zenoh中的发现，以便简化想要利用集团（clique）连接性的系统的部署，在多播不可用或不希望使用多播的情况下，现在支持基于Closure的发现。换句话说，从单个对等方开始，可以发现其 Closure，或者换句话说，可以从此起点直接或间接地到达对等点。

**基于区域（Region）的路由**。这实际上意味着两件事：（1）建立和维护路由表随区域大小缩放所需的路由信息，以及（2）每个区域都可以决定是否要在任意连接图上进行路由的集团。通过允许在区域内使用最合适的路由技术，这种路由方法将极大地提高可伸缩性和性能。

### 2.3 移植方案设计

软件移植大体可以分为两个核心步骤：**如何构建**和**如何处理源码以兼容目标平台**。构建部分主要涉及交叉编译工具链的选择和配置、构建系统的调整（如 CMake 文件修改）以及目标平台的编译环境搭建，确保代码能够在目标操作系统（如 SylixOS）上生成可执行文件或库。而源码处理则需要分析目标平台的特性，对代码进行必要的调整，例如替换或屏蔽与 SylixOS 不兼容的系统调用、重写依赖于特定操作系统的逻辑，以及针对目标平台的库和接口提供适配层，最终确保程序功能和性能的正确性与稳定性。

Zenoh-Pico 基于 CMake 工程组织代码，通过利用 SylixOS 提供的交叉编译工具链，开发者可以配置目标平台的编译参数，快速构建出适用于 SylixOS 的项目工程。CMake 的模块化设计不仅简化了交叉编译流程，还支持对不同平台的特性进行精细化适配，从而最大程度地减少对源代码的修改。

本项目依照《SylixOS-软件移植手册》，基于Msys2构建CMake工程。主要步骤如下：

**STEP1**：搭建交叉编译环境，按照相关文档配置好 CMake 和 MSYS2 工具，确保交叉编译工具链能够正确识别 SylixOS 的目标架构。此外，需要编写 SylixOS 平台专用的 sylixos.toolchain.cmake 交叉编译配置文件。

**STEP2**：导入 zenoh-pico-main 的源代码，启动编译流程，并对编译过程中出现的错误逐一分析和修改。常见的问题可能包括目标平台与源代码中使用的系统调用或库的不兼容、依赖项缺失、以及与 SylixOS 平台特性相关的调整。

**STEP3**：生成目标库文件后，构建一个或多个测试程序，用于验证库文件的功能与性能。通过将测试程序部署到 SylixOS 平台，运行并观察实际的运行情况，验证项目的正确性与稳定性。

Zenoh-pico项目版本和SylixOS平台版本情况如下：

Zenoh-pico项目版本：1.1.0

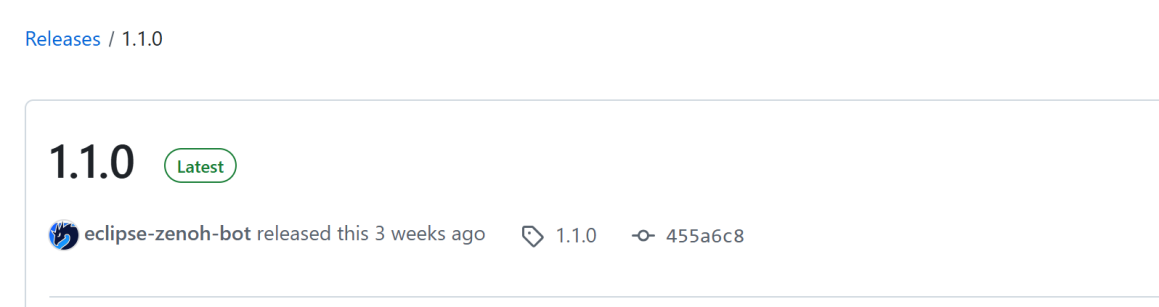


图3 移植的Zenoh-pico项目版本

sylixOS平台版本：



图4 SylixOS版本

### 2.4 Zenoh-pico模块剖析

Zenoh-pico编译的最终目标是生成libzenohpico.so，如图5所示：

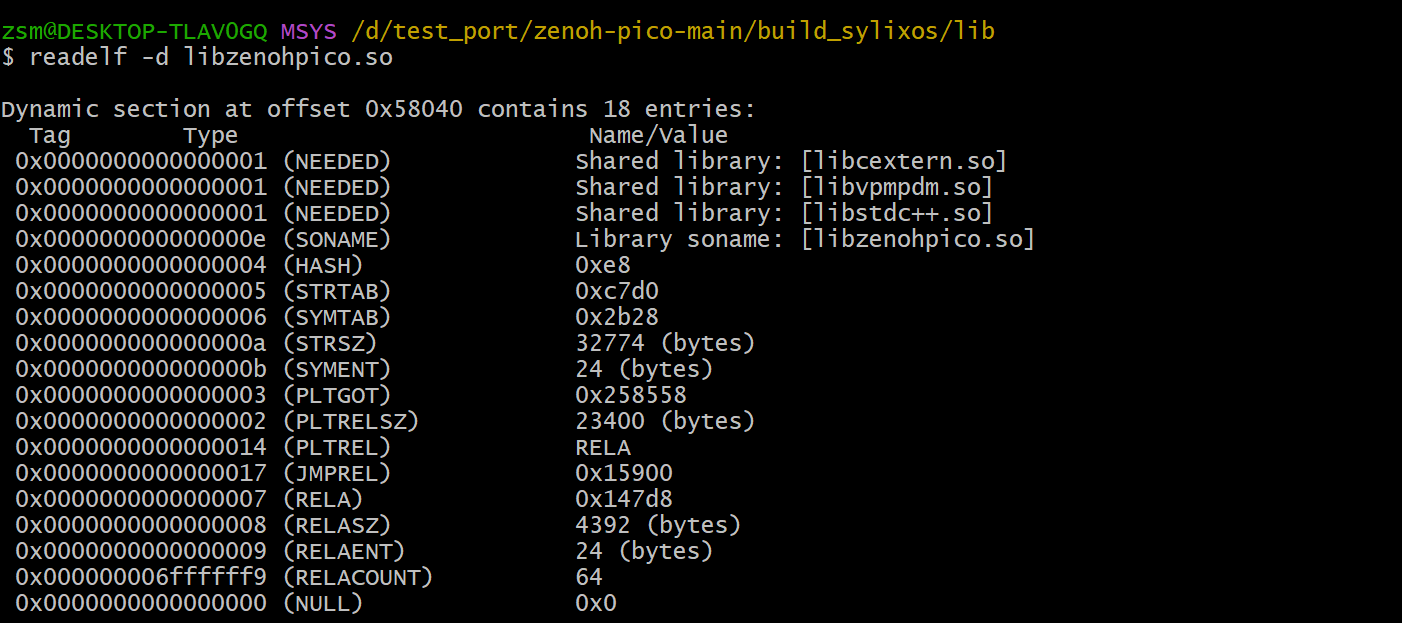


图5 libzenohpico.so的依赖库信息

libvpmPdm.so是一个具体的依赖库，可能与平台相关功能（如消息处理、协议栈等）有关。libstdc++.so表示 libzenohpico.so 依赖于标准 C++ 库。这表明 Zenoh-Pico 使用了 C++ 语言的一些特性。

下图中是构建的示例程序与测试用例。

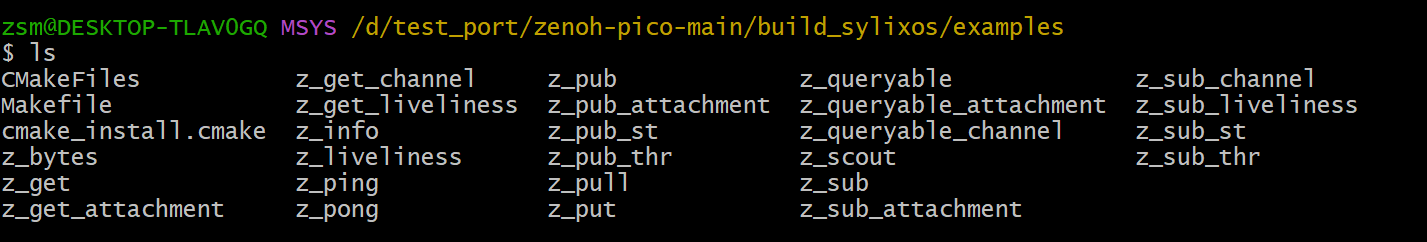


图6 示例程序

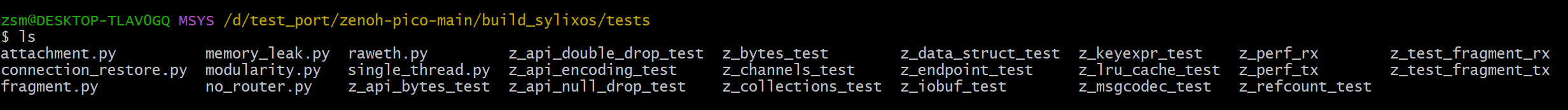


图7 测试用例

## 第三章 目标完成情况

截至本文档攥写完成，依照前文的项目目标规划，最终基本实现了项目移植目标，成功将 Zenoh-pico 移植到 SylixOS 平台，确保其核心功能（如数据发布、订阅、查询和数据存储）在 SylixOS 上能够正常运行，Zenoh-pico 的所有示例程序和测试用例在 SylixOS 上均可编译、运行并通过测试，确保核心功能的完整性和稳定性。具体情况如表1所示。

表1 目标完成情况

|  |  |  |
| --- | --- | --- |
| **程序名** | **目标功能** | **完成情况** |
| z\_api\_bytes\_test | 测试zenoh-pico 库在数据操作、序列化管理、内存控制等方面的功能 | 完成，测试通过 |
| z\_api\_double\_  drop\_test | 测试zenoh-pico库在键表达式和配置管理等方面的功能 | 完成，测试通过 |
| z\_api\_encoding\_test | 测试zenoh-pico库中编码功能的基本操作 | 完成，测试通过 |
| z\_api\_null\_drop\_test | 测试zenoh-pico库中的多种数据类型 | 完成，测试通过 |
| z\_bytes\_test | 测试zenoh-pico 中字节集合的核心功能 | 完成，测试通过 |
| z\_channels\_test | 测试FIFO通道和环形通道的功能 | 完成，测试通过 |
| z\_collections\_test | 测试zenoh-pico 的集合数据结构 | 完成，测试通过 |
| z\_data\_struct\_test | 测试zenoh-pico 中常见的数据结构操作、内存管理和接口等功能 | 完成，测试通过 |
| z\_endpoint\_test | 测试zenoh-pico库中locator和endpoint功能 | 完成，测试通过 |
| z\_iobuf\_test | 测试zenoh-pico的缓冲区结构和I/O切片功能 | 完成，测试通过 |
| z\_keyexpr\_test | 测试键表达式（Key Expression）的各种操作 | 完成，测试通过 |
| z\_lru\_cache\_test | 测试一个自定义类型的LRU Cache功能 | 完成，测试通过 |
| z\_msgcode\_test | 测试zenoh-pico 协议栈中各种消息类型的编解码功能 | 完成，测试通过 |
| z\_refcount\_test | 测试zenoh-pico中的引用计数功能 | 完成，测试通过 |
| z\_perf\_rx | 测试zenoh-pico的订阅、数据接收和统计功能，并对不同的测试条件进行了相应的处理 | 测试失败，详见5.1 |
| z\_perf\_tx | 测试zenoh-pico的发布、数据包发送、会话管理和网络配置等功能 | 完成，测试通过 |
| z\_test\_fragment\_rx | 测试 zenoh-pico 的订阅功能、数据接收与处理、会话管理、网络配置以及程序的终止控制逻辑 | 测试失败，详见5.1 |
| z\_test\_fragment\_tx | 测试 zenoh-pico 的发布功能、会话管理、网络配置、数据发布选项设置和内存管理等关键功能 | 完成，测试通过 |
| z\_bytes | 测试zenoh-pico的字节处理、序列化与反序列化、复合数据类型支持、以及切片迭代器的功能 | 完成，测试通过 |
| z\_get | 通过zenoh协议进行查询操作 | 完成，测试通过 |
| z\_get\_attachment | 通过zenoh协议发送查询请求并处理带有附件和编码信息的回复 | 完成，测试通过 |
| z\_get\_channel | 使用环形通道发送查询并接收处理回复 | 完成，测试通过 |
| z\_get\_liveliness | 使用zenoh pico的活跃性查询功能检测网络中活跃的资源 | 完成，测试通过 |
| z\_info | 获取网络中自身设备、路由器和对等节点的标识符 | 完成，测试通过 |
| z\_liveliness | 实现创建、声明和撤销生存性令牌的完整流程 | 完成，测试通过 |
| z\_ping | 测试发布和订阅的基础功能 | 完成，测试通过 |
| z\_pong | 处理来自 test/ping 路径的消息，并将其作为响应转发到 test/pong 路径 | 完成，测试通过 |
| z\_pub | 实现了一个 zenoh pico 的发布者程序 | 完成，测试通过 |
| z\_pub\_attachment | 测试附加元数据、编码、时间戳等操作 | 完成，测试通过 |
| z\_pub\_st | 通过zenoh发布数据，并且能够在特定的连接和监听模式下运行 | 完成，测试通过 |
| z\_pub\_thr | 不断发布数据，并通过zenoh进行流式通信 | 完成，测试通过 |
| z\_pull | 通过指定的 key expression 定期拉取数据，处理数据并输出 key-value 对 | 完成，测试通过 |
| z\_put | 通过zenoh pico库向指定的键表达式发送数据 | 完成，测试通过 |
| z\_queryable | 创建一个可查询的服务 | 完成，测试通过 |
| z\_queryable  \_attachment | 处理带有附件的查询请求 | 完成，测试通过 |
| z\_queryable  \_channel | 创建一个支持查询的通道 | 完成，测试通过 |
| z\_scout | 测试寻找网络中的Zenoh节点，并通过回调处理返回的Hello消息 | 完成，测试通过 |
| z\_sub | 声明一个订阅者，接收指定主题的消息，并处理收到的数据 | 完成，测试通过 |
| z\_sub\_attachment | 处理带有附件的消息，使用反序列化技术解析附件内容，输出键值对信息以及消息的其他元数据 | 完成，测试通过 |
| z\_sub\_channel | 客户端订阅特定的主题表达式，并通过FIFO通道接收消息 | 完成，测试通过 |
| z\_sub\_liveliness | 客户端订阅特定主题，并根据收到的更新打印存活的令牌或被删除的令牌 | 完成，测试通过 |
| z\_sub\_st | 客户端订阅指定主题并接收消息 | 完成，测试通过 |
| z\_sub\_thr | 测试使用多线程订阅指定主题并统计接收到的消息数量 | 完成，测试通过 |

## 第四章 移植过程

在前面几章中，我们已经明确了需要移植的项目以及目标系统SylixOS的大致内容。本章将在此基础上详细阐述移植项目所使用的方法以及移植过程。

### 4.1 SylixOS交叉编译

为了方便将第三方软件移植到SylixOS上，除了直接使用RealEvo-IDE工程构建外，对于复杂项目，采用其原有构建方式更为简洁高效。这里需要基于交叉编译环境MSYS2来实现。

#### 4.1.1 MSYS2环境搭建

MSYS（MinimalGNU（POSIX）system on Windows）是一种在Windows操作系统上运行类似于 Unix操作系统的环境的软件工具。它旨在为开发者提供在Windows平台上进行软件开发时更接近Unix/Linux环境的体验。

1. **配置msys2基本内容**

安装msys2，并修改安装目录下的msys2.ini文件，将MSYS\_PATH\_TYPE=inherit的注释打开，使得MSYS2支持全局环境变量

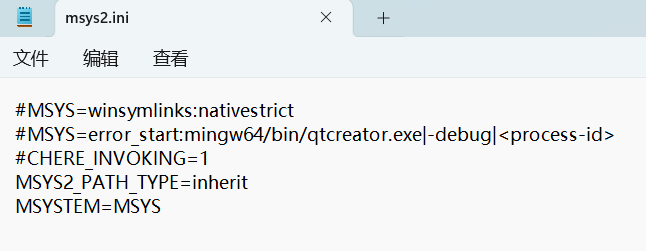


图8 MSYS配置

1. **配置msys2编译环境**

打开msys2.exe，在MSYS2终端输入以下命令安装编译相关环境，过程中会出现2次选择，直接敲击回车选择默认即可。接着会出现1次是否安装的选项，需要手动输入Y，开始安装。如图所示：

pacman -S mingw-w64-x86\_64-toolchain mingw-w64-i686-toolchain base-devel gcc make base-devel

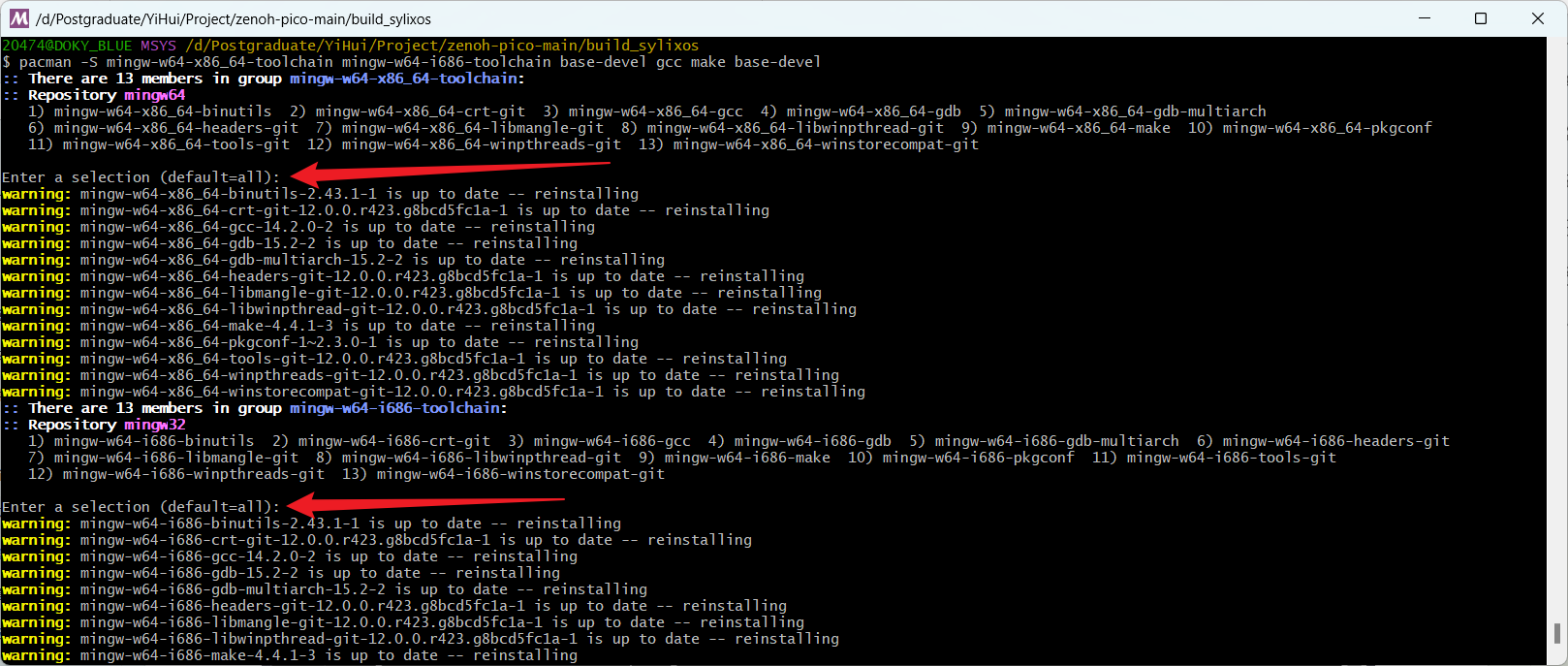


图9 安装MSYS2编译环境

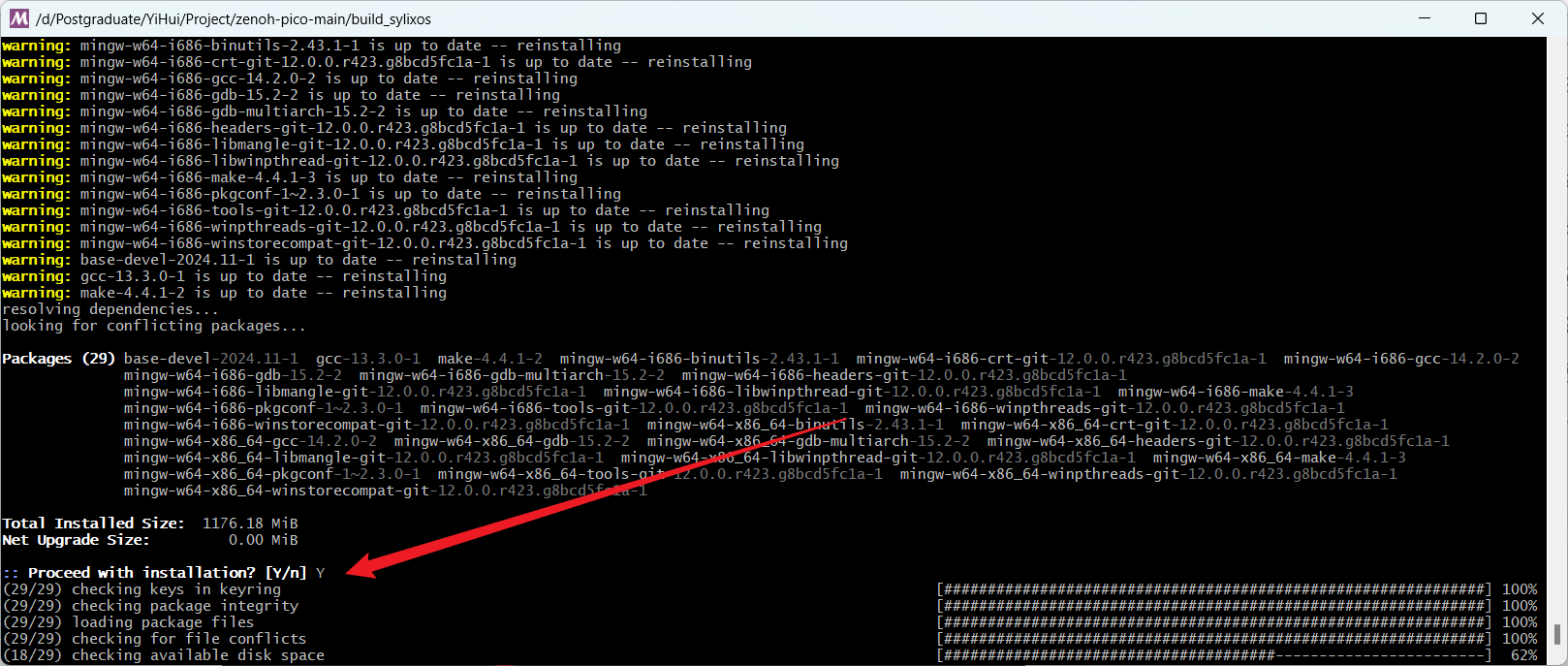


图10 安装MSYS2编译环境

#### 4.1.2 交叉编译环境搭建

1. **cmake环境搭建**

安装cmake，并在cmake安装目录下的/share/cmake-3.27.1/Modules/ Platform文件夹中添加SylixOS.cmake、SylixOS-GNU.cmake、SylixOS-GNU-C.cmake、SylixOS-GNU-CXX.cmake、SylixOS-GNU-Fortran.cmake五个文件，以使得cmake支持SylixOS系统。

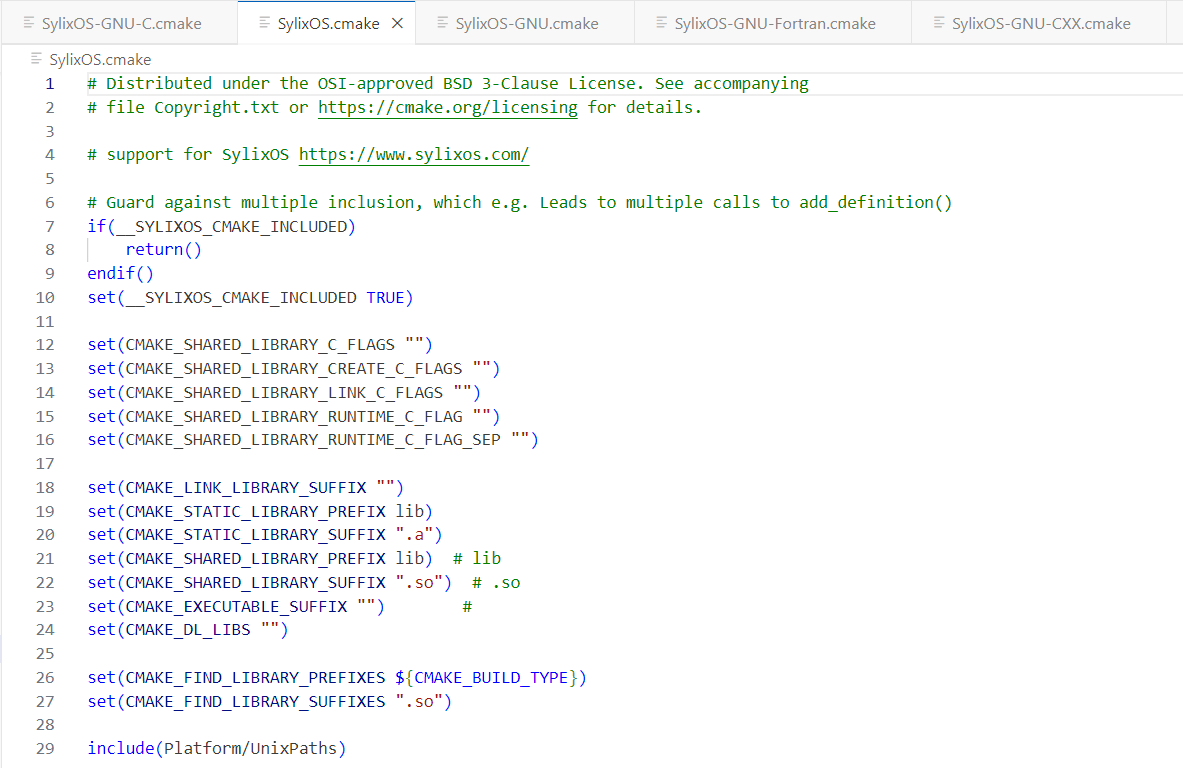


图11 配置cmake环境

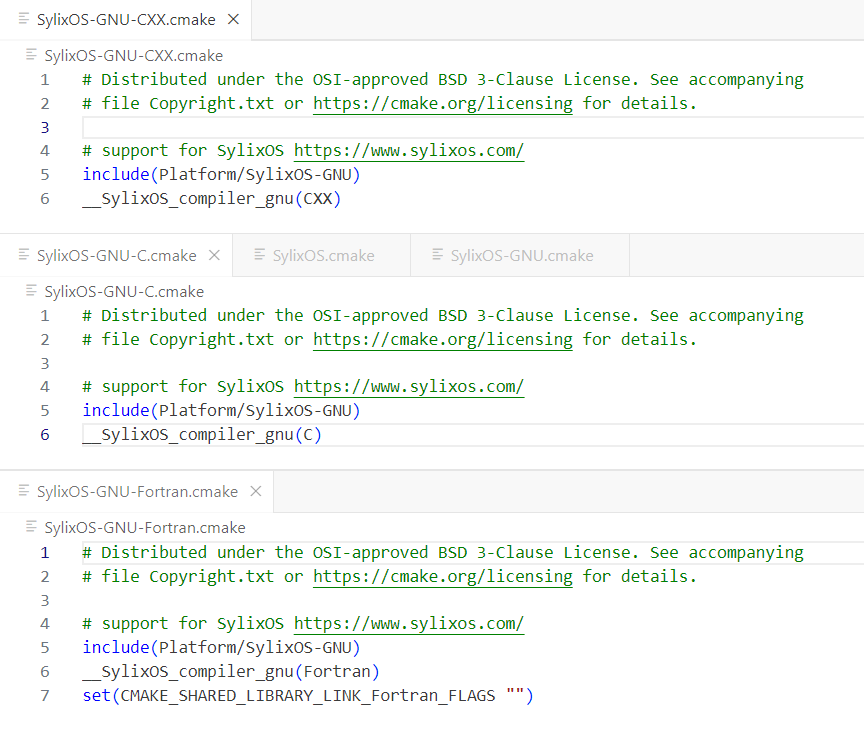


图12 配置cmake环境

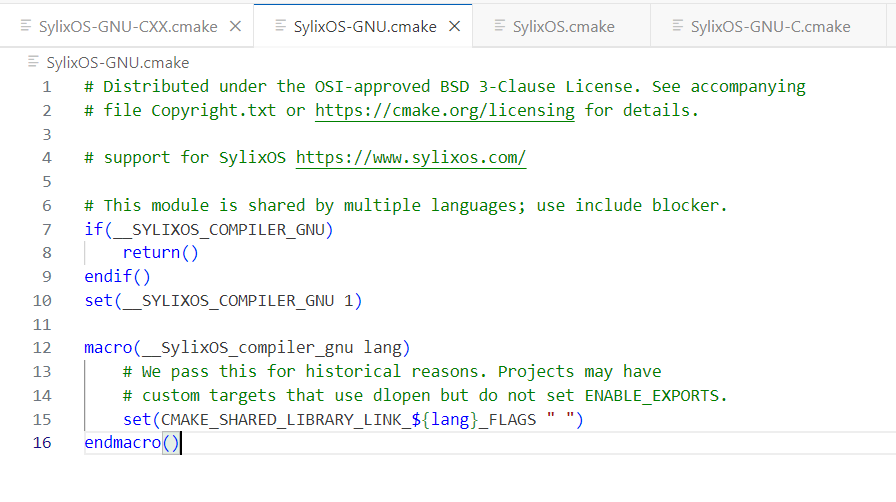


图13 配置cmake环境

1. **交叉编译环境搭建**

CMake提供了CMAKE\_TOOLCHAIN\_FILE变量，通过编写CMAKE\_TOOLCHAI N\_FILE文件，可以达到不同平台之间交叉编译的目的。首先从zenoh-pico的github上下载最新版本的源码，打开msys2.exe进入源码文件夹中，创建sylixos\_toolchain.cmake文件。

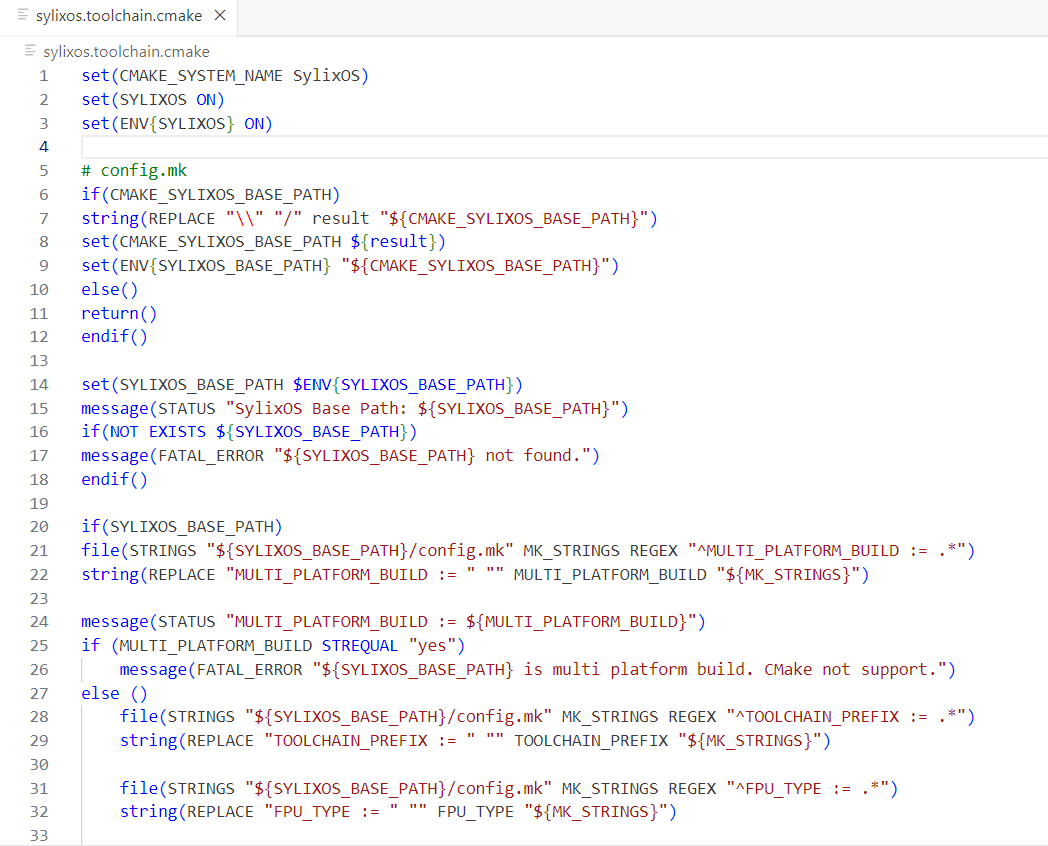


图14 配置sylixos.toolchain.cmake文件

### 4.2 编译zenoh-pico

在完成MSYS2环境搭建和SylixOS交叉编译环境搭建后，就可以对第三方软件进行交叉编译实现移植了。

#### 4.2.1 构建CMake

在项目目录下执行以下指令，创建一个新的构建文件夹并切换到构建文件夹

mkdir build\_sylixos && cd build\_sylixos

在构建文件夹中执行以下指令进行CMake的构建  
cmake -G "MSYS Makefiles"

-DCMAKE\_VERBOSE\_MAKEFILE=TRUE

-DCMAKE\_MAKE\_PROGRAM="make"

-DCMAKE\_VERBOSE\_MAKEFILE=ON

-DCMAKE\_BUILD\_TYPE=Debug

-DCMAKE\_SYLIXOS\_BASE\_PATH="D:/Postgraduate/YiHui/Project/VSOA\_Project/x86\_base"

-DCMAKE\_INSTALL\_PREFIX="\_install"

-DCMAKE\_TOOLCHAIN\_FILE="../sylixos.toolchain.cmake"

-DBUILD\_TESTING=ON -Wno-dev ../

其中，参数说明如下：

-G "MSYS Makefiles"：生成MSYS Makefiles格式，具体可使用cmake --help查看。

-DCMAKE\_MAKE\_PROGRAM：设置make工具，可以设置绝对路径或相对路径。

-DCMAKE\_VERBOSE\_MAKEFILE：设置是否打印编译信息，ON or OFF。

-DCMAKE\_BUILD\_TYPE：设置编译版本，Release or Debug。

-DCMAKE\_SYLIXOS\_BASE\_PATH：设置SylixOS Base路径。

-DCMAKE\_INSTALL\_PREFIX=：设置安装编译后安装目录。

-DDCMAKE\_TOOLCHAIN\_FILE：设置交叉编译文件路径。绝对路径或相对路径。

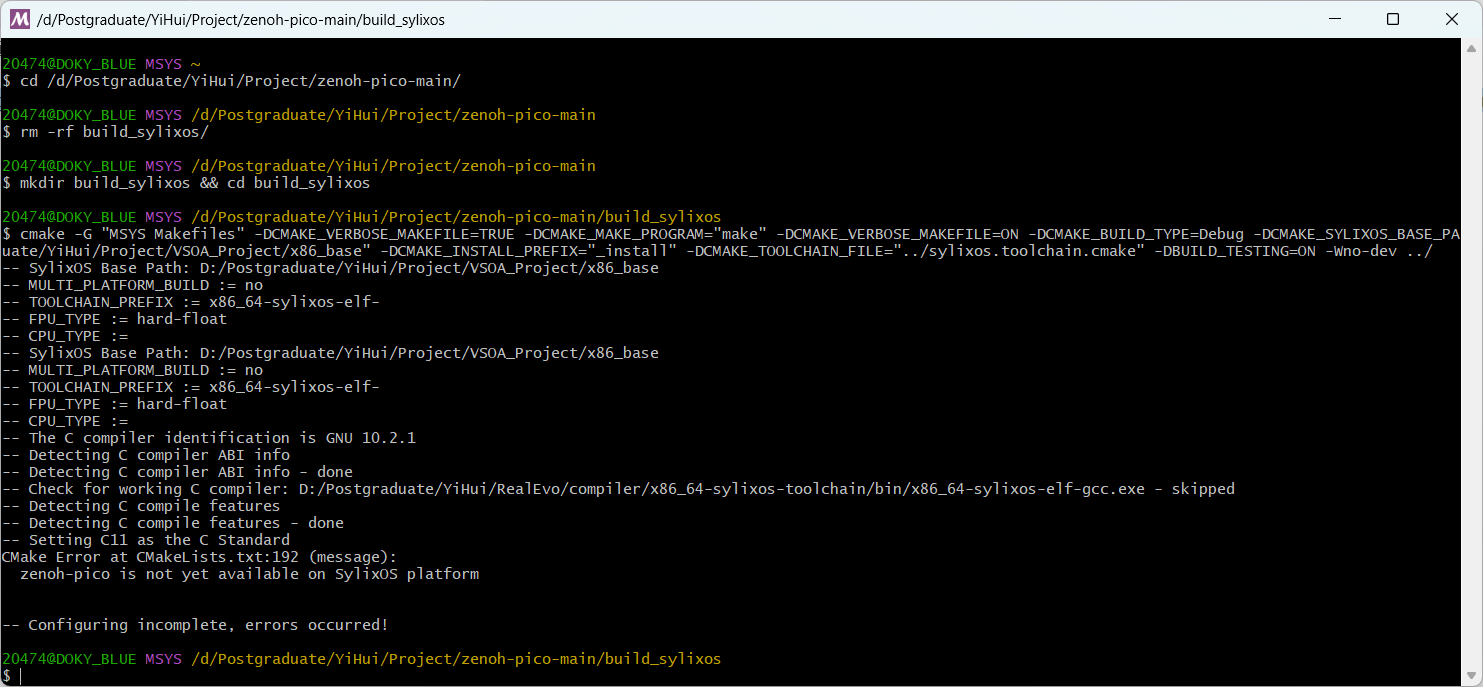
构建过程遇到报错  


图15 构建CMake过程报错

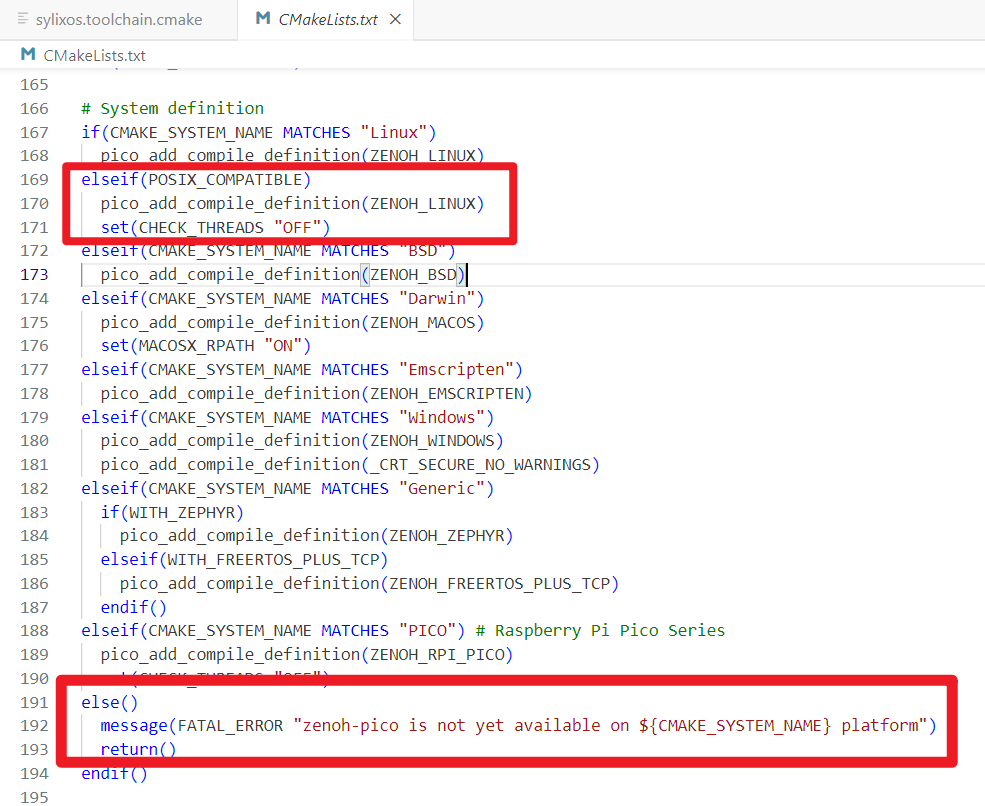
错误提示zenoh-pico不支持SylixOS平台，查看CMakeLists.txt内容如下  


图16 查询zenoh-pico的CMakeLists.txt文件

虽然zenoh-pico的编译平台没有SylixOS，但是SylixOS兼容POSIX，在sylixos.toolc hain.cmake中添加以下内容即可。

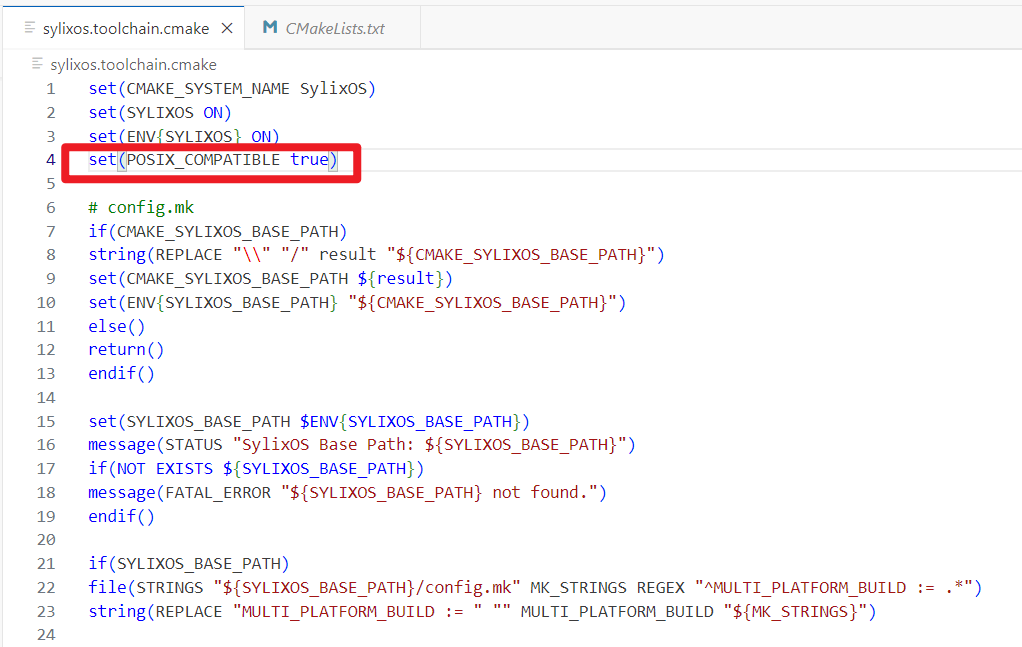


图17 修改sylixos.toolchain.cmake文件

再次构建cmake，成功构建

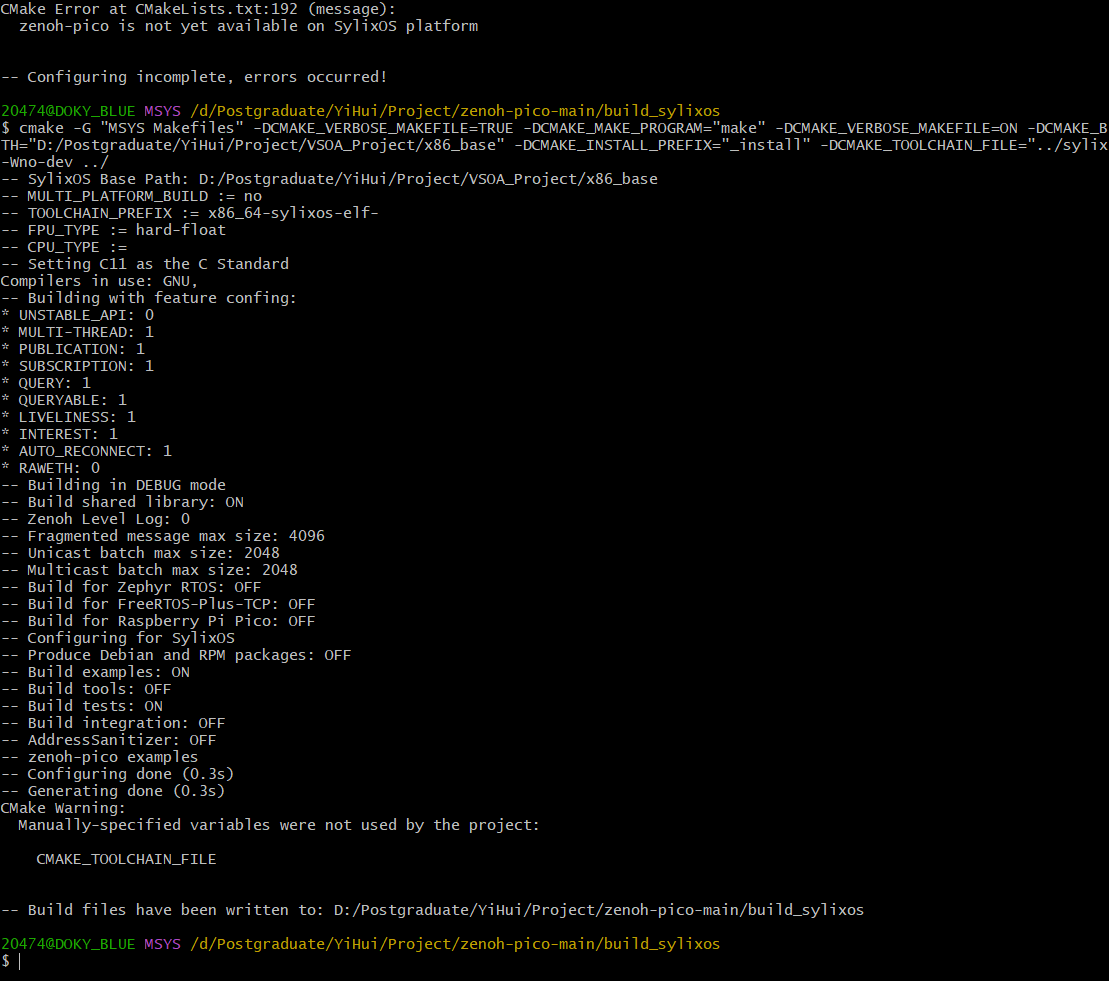


图18 成功构建CMake

#### 4.2.2 编译工程

构建cmake后执行make来编译项目，编译过程遇到报错如下：

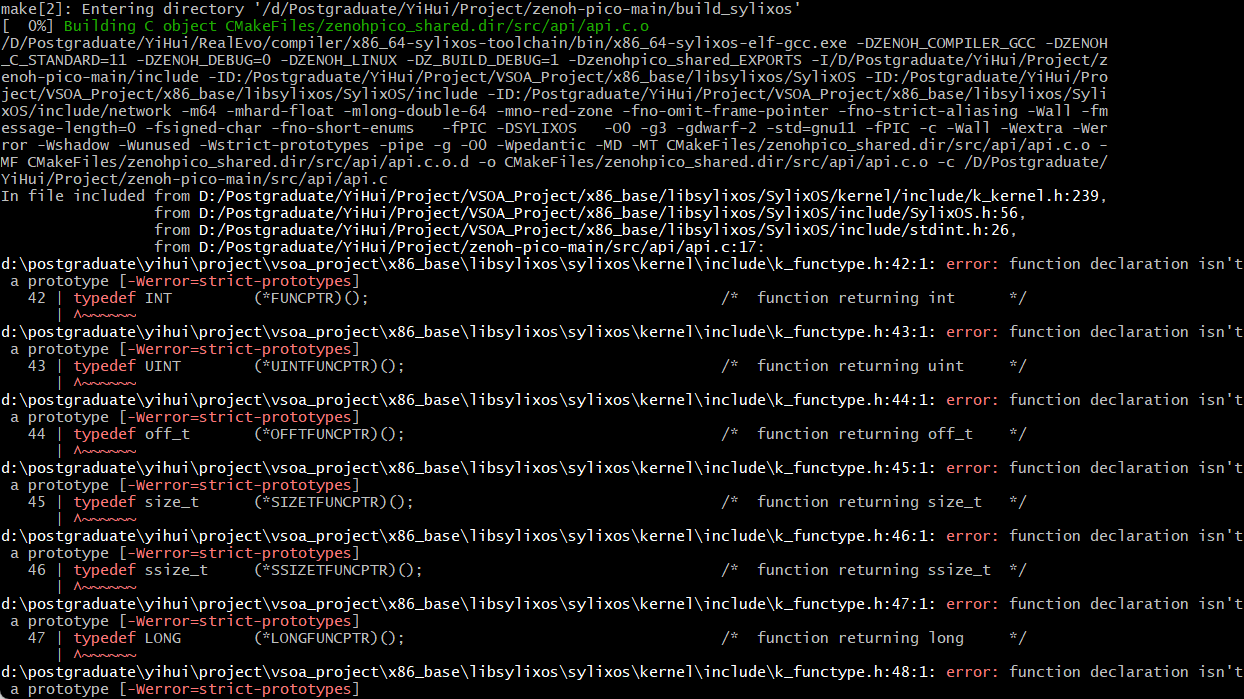


图19 编译过程报错

经排查发现是由于编译器选项 -Werror=strict-prototypes将警告function declaration isn't a prototype转化为错误导致的。这种警告源于函数声明中未明确指定参数为(void)，而是使用了空括号()。该警告不会影响具体软件功能，遂查询CMakeLists.txt中关于Werror的内容，将-Werror的编译器选项删除。

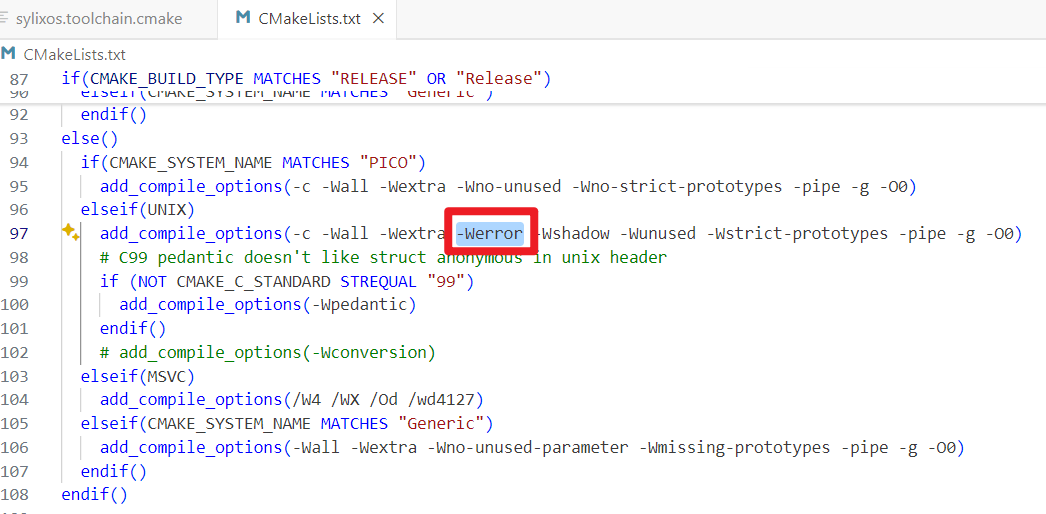


图20 修改CMakelists.txt文件

再次对项目进行编译，编译过程遇到报错如下

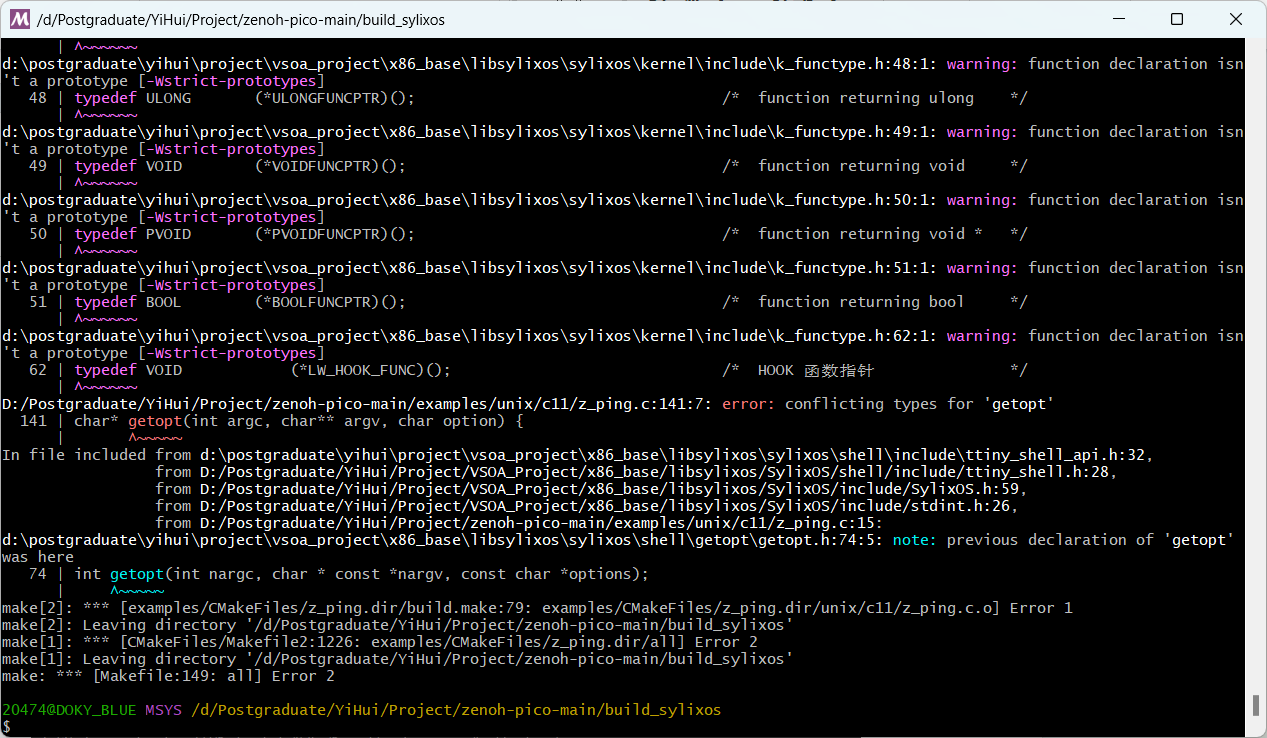


图21 编译过程报错

经排查后发现该报错是因为在/example/unix/c11/z\_ping.c中的函数char\* getopt(int argc, char\*\* argv, char option)，和/x86\_base/libsylixos/sylixos/shell/getopt/getopt.h中的函数int getopt(int nargc, char \* const \*nargv, const char \*options) 定义冲突，则将z\_ping.c下的函数重命名为sylix\_getopt即可，再次编译后通过。

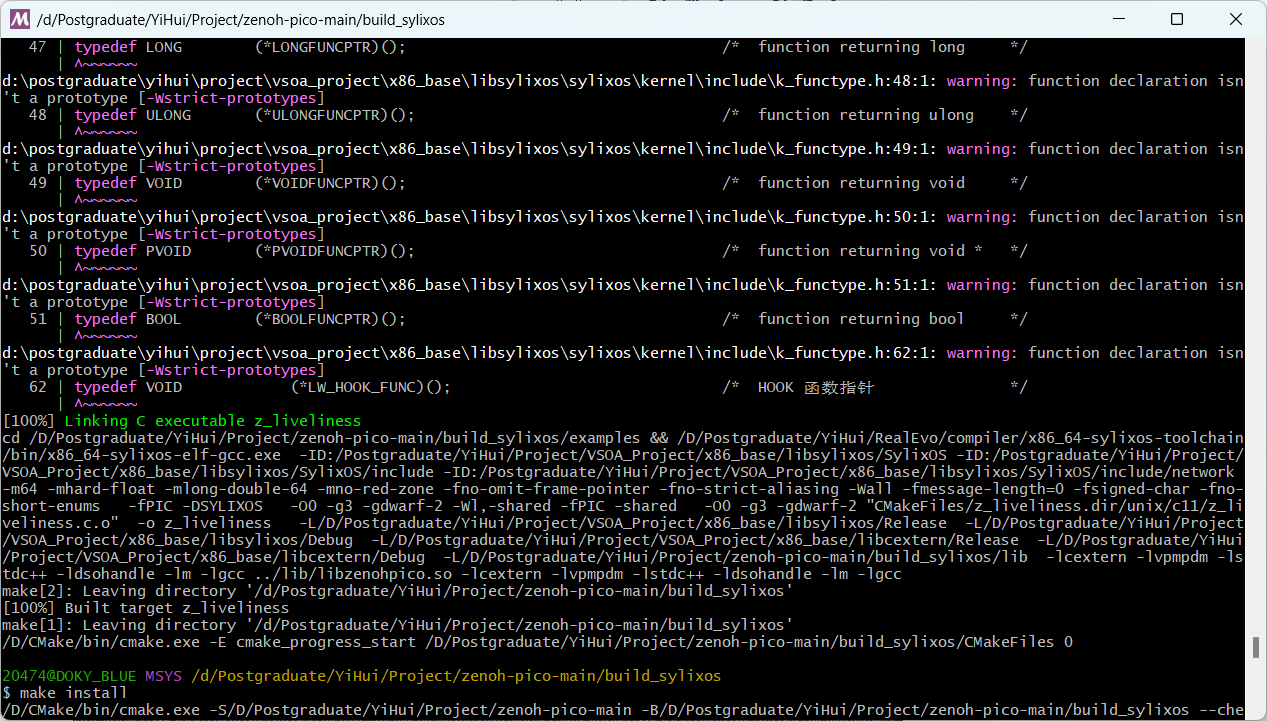


图22 编译通过

编译通过后进行安装，执行make install。安装过程无报错，在安装完毕后将build\_sylixos文件夹的内容上传至sylixos虚拟机中进行测试和验证即可。

## 第五章 测试与验证

zenoh-pico项目中提供了许多测试用例和运行示例，包含了各种api接口测试、单元测试以及各种情况下的集成测试，通过这些测试和示例我们可以确定我们交叉编译后的结果是否存在问题，以此确认最终移植结果。

编译过后共生成以下test和example可执行文件

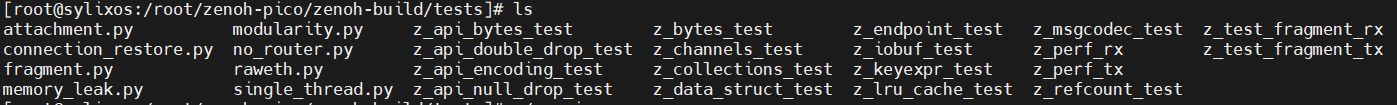


图23 生成的test文件

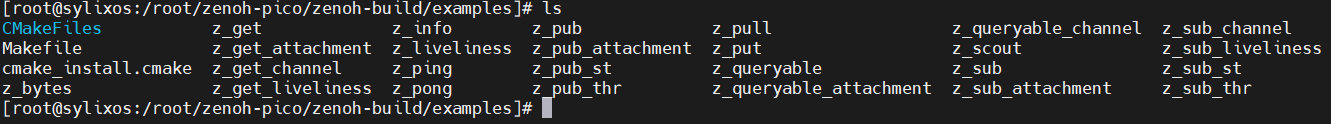


图24 生成的example文件

**Test：**z\_api\_bytes\_test、z\_bytes\_test、z\_endpoint\_test、z\_msgcodec\_test、z\_test\_fragment\_rx、z\_api\_double\_drop\_test、z\_channels\_test、z\_iobuf\_test、z\_perf\_rx、z\_test\_fragment\_tx、z\_api\_encoding\_test、z\_collections\_test、z\_keyexpr\_test、z\_perf\_tx、z\_api\_null\_drop\_test z\_data\_struct\_test、z\_lru\_cache\_test、z\_refcount\_test

**Example：**z\_get、z\_info、z\_pub、z\_pull、z\_queryable\_channel、z\_sub\_channel、z\_get\_attachment、z\_liveliness、z\_pub\_attachment、z\_put、z\_scout、z\_sub\_liveliness、z\_get\_channel、z\_ping z\_pub\_st、z\_queryable、z\_sub、z\_sub\_st、z\_bytes、、z\_get\_liveliness、z\_pong、z\_pub\_thr、z\_queryable\_attachment、z\_sub\_attachment、z\_sub\_thr

### 5.1 功能测试

zenoh-pico是Zenoh项目使用C代码的一个轻量级实现，主要用于数据分发、查询、存储等功能，是一个分布式中间件。其主要API接口有以下部分：

1. 字节流和序列化接口：z\_bytes\_reader\_seek()ze\_serializer\_serialize\_bool()等
2. 配置和对象创建接口：z\_keyexpr\_from\_str()、z\_config\_default()等
3. 编码处理接口：z\_internal\_encoding\_null()、z\_encoding\_from\_str()等

本节剩余部分将逐步对编译过程生成的可执行测试文件进行测试，可执行文件均以assert断言的形式判断是否出错，若运行可执行文件后不输出错误则证明测试通过。

1. **z\_api\_bytes\_test：**测试了 zenoh-pico 库在数据操作、序列化管理、内存控制等方面的功能
2. **z\_api\_double\_drop\_test：**测试了zenoh-pico库在键表达式和配置管理等方面的功能
3. **z\_api\_encoding\_test：**测试了zenoh-pico库中编码功能的基本操作，包括编码对象的创建、转换、与模式的关联、常量值的使用以及编码对象之间的比较
4. **z\_api\_null\_drop\_test：**通过宏定义的方式，对zenoh-pico库中的多种数据类型进行了测试，重点验证了对象的初始化、无效值处理、资源释放（包括双重释放的安全性）以及对象所有权转移等方面的功能

测试可执行文件执行结果如下，测试全部通过

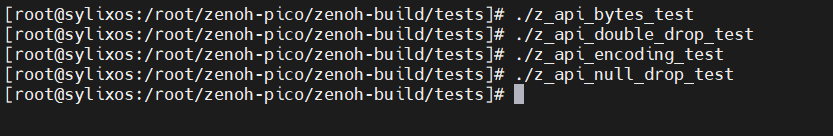


图25 部分测试内容

1. **z\_bytes\_test：测**试了 zenoh-pico 中字节集合的核心功能，包括创建、读取、写入、追加和随机访问等功能
2. **z\_channels\_test：**测试了FIFO通道和环形通道的功能，包括消息发送、接收以及异常条件处理等功能。
3. **z\_collections\_test：**对 zenoh-pico 的集合数据结构（如环形缓冲区、栈（LIFO）、队列（FIFO）、以及键值映射）的测试。
4. **z\_data\_struct\_test：**测试了 zenoh-pico 中常见的数据结构操作、内存管理和接口的核心功能。
5. **z\_endpoint\_test：**测试了 zenoh-pico 库中 locator 和 endpoint 的功能，验证了相关函数对定位符（locator）和端点（endpoint）字符串的解析、验证、以及内部数据结构的操作是否正确。
6. **z\_iobuf\_test：**测试zenoh-pico的缓冲区结构和I/O切片功能，通过断言来验证缓冲区的读写行为、可扩展性和数据一致性。
7. **z\_keyexpr\_test：**测试了键表达式（Key Expression）的各种操作，包括交集（intersect）、包含关系（includes）、标准化（canonize）、相等性（equals）等。
8. **z\_lru\_cache\_test：**测试了一个自定义类型的“最近最少使用缓存（LRU Cache）”功能，其核心功能是定义和测试一个容量固定的缓存，其中数据项会根据访问历史动态调整。
9. **z\_msgcode\_test：**通过生成随机的消息数据，并对这些数据进行编码和解码，测试了 zenoh-pico 协议栈中各种消息类型的编解码功能。
10. **z\_refcount\_test**：测试了zenoh-pico中的引用计数功能，特别是对对象的引用计数、克隆、弱引用等操作

测试可执行文件部分执行结果如下，测试全部通过

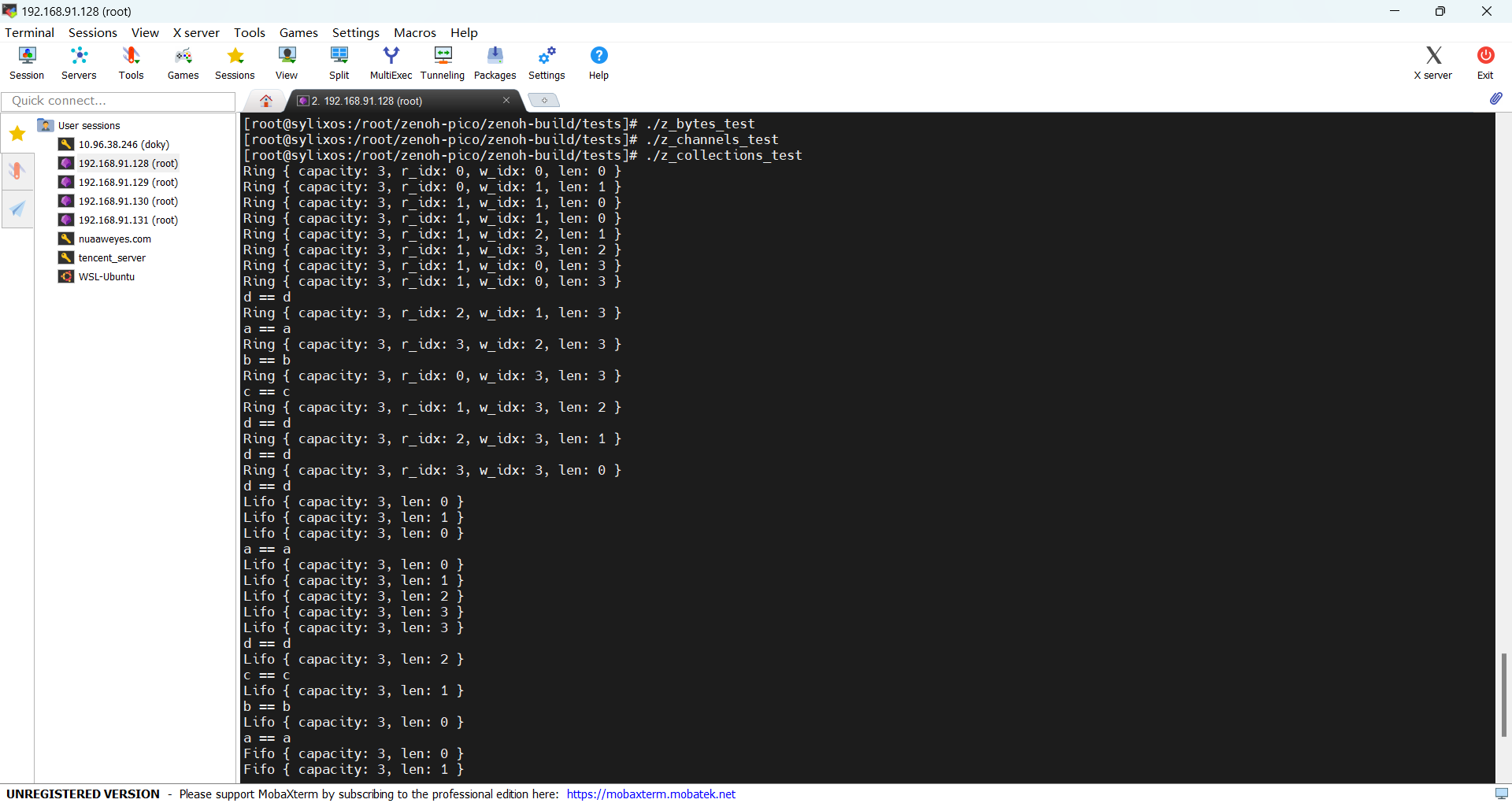


图26 部分测试内容

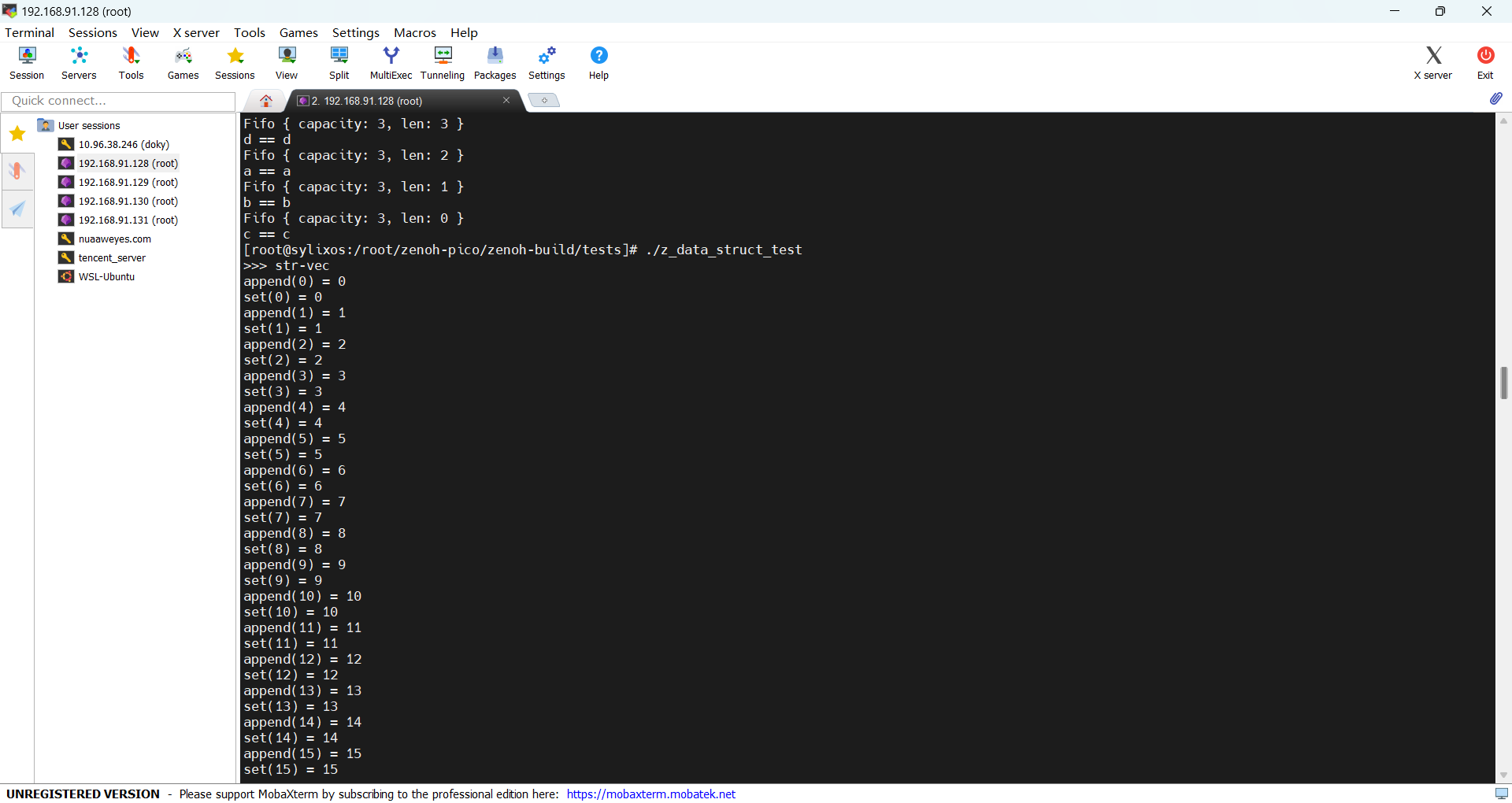


图27 部分测试内容

### 5.2 通信测试

除功能测试之外，zenoh-pico提供了两对用于通信的测试用例，通过这些用例可以测试在多端之间借助路由器进行通信的功能，具体测试过程如下：

在Linux虚拟机上启动zenoh路由器，记录路由器的IP地址，并写入测试用例中。打开两个SylixOS的虚拟机，分别执行接收数据的测试用例和发送数据的测试用例，若两个SylixOS虚拟机成功发送/接收数据，则证明测试通过。

1. **z\_perf\_rx：**进行网络测试，主要测试了zenoh-pico的订阅、数据接收和统计功能，并对不同的测试条件进行了相应的处理，与z\_perf\_tx配套执行。
2. **z\_perf\_tx：**进行网络测试，主要测试了zenoh-pico的发布、数据包发送、会话管理和网络配置等功能，与z\_perf\_rx配套执行。

该对测试用例执行结果如下，测试失败

存在问题：测试的时候发现该部分数据包发送时是从1048576大小开始依次往下递减发送的，接收时只能从1024开始接收，该部分存在问题。

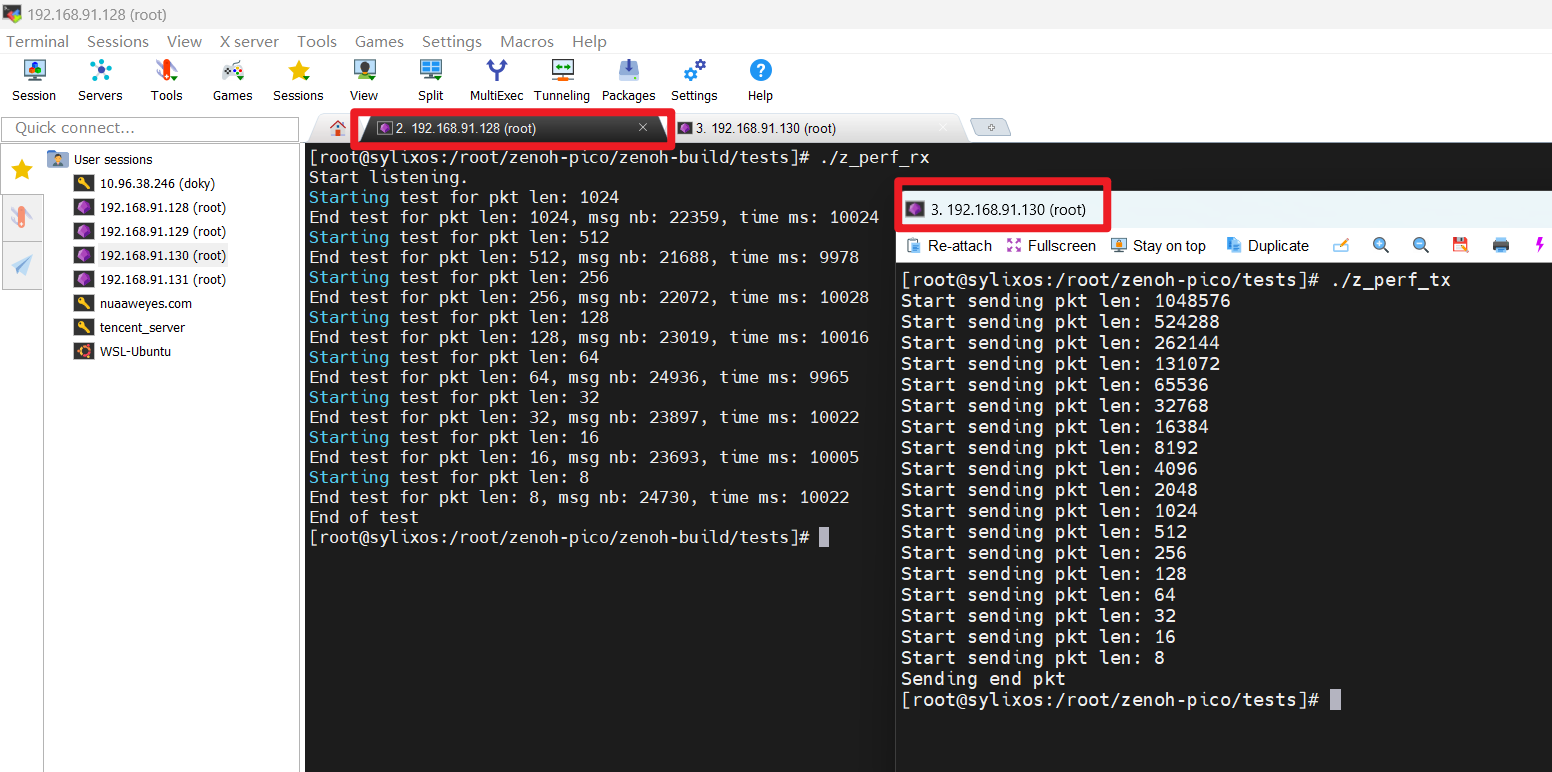


图28 z\_perf\_tx和z\_perf\_rx测试案例

1. **z\_test\_fragment\_rx：**测试 zenoh-pico 的订阅功能、数据接收与处理、会话管理、网络配置以及程序的终止控制逻辑。
2. **z\_test\_fragment\_tx：**测试 zenoh-pico 的发布功能、会话管理、网络配置、数据发布选项设置和内存管理等关键功能。

该对测试用例执行结果如下，测试失败

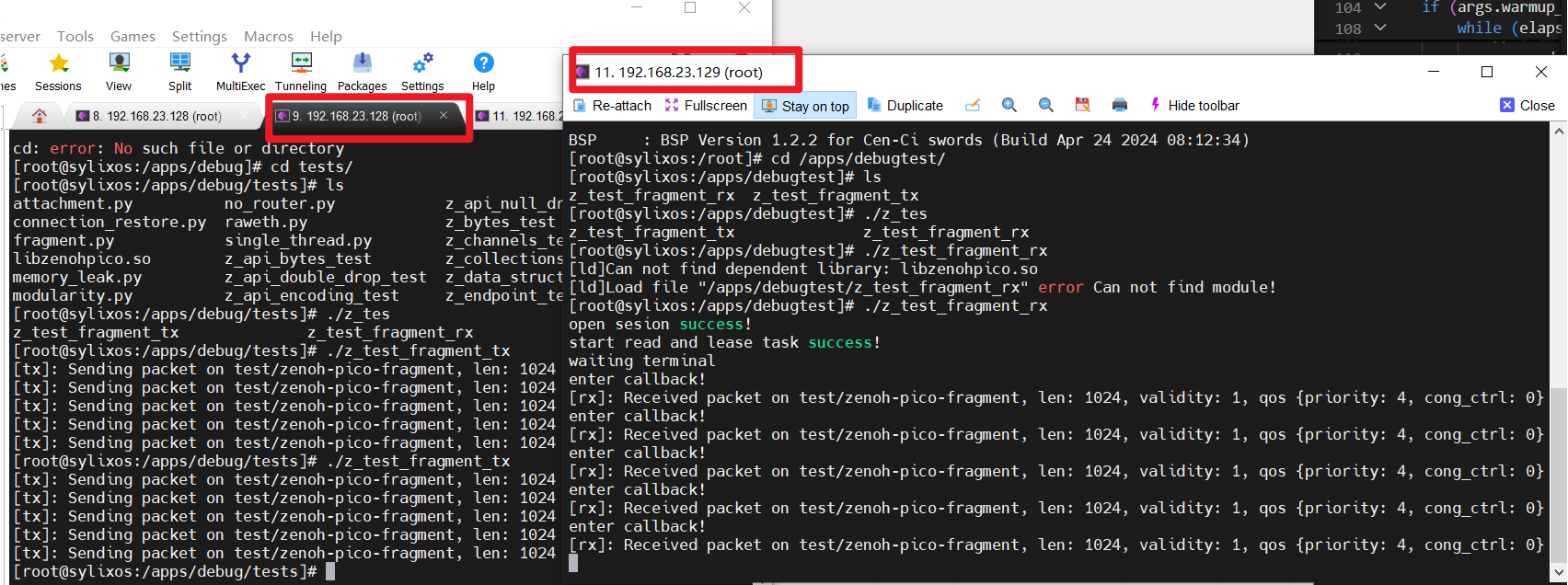
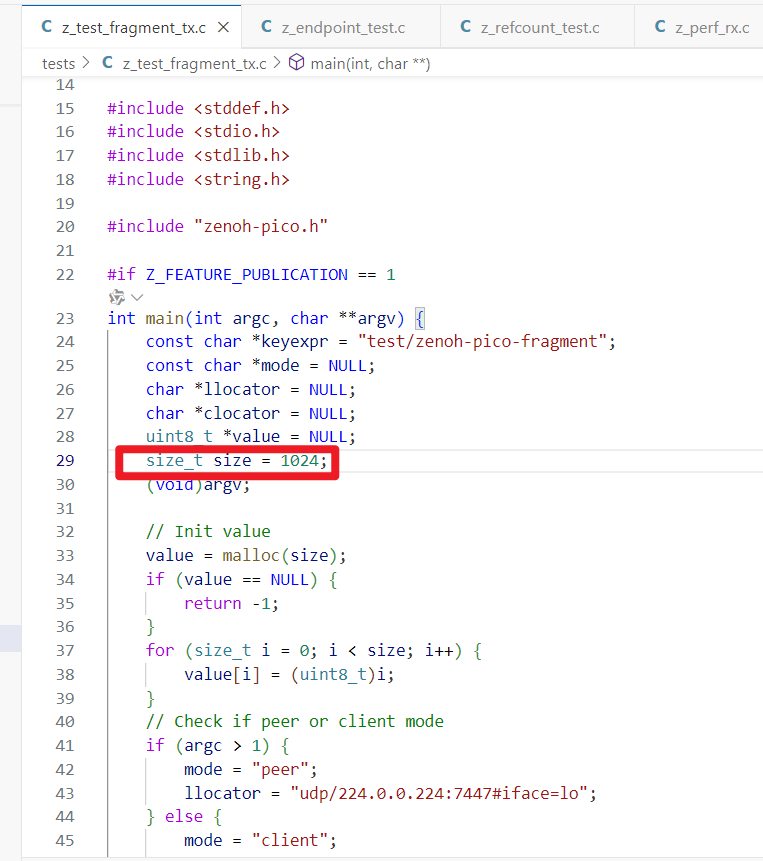


图29 部分测试内容

存在问题：在测试时发现与前一对测试用例同样的问题：初始时发送的数据包为10000，接收方无法接收，将发送数据包修改为1024时才能接收到。

图30 修改发送数据包大小

### 5.3 验证示例

除了测试用例之外，zenoh-pico同样提供了许多运行示例，这些示例模拟了在各种情况下的zenoh-pico运行场景，通过这些示例，可以更加准确的确定移植结果。

1. **z\_bytes：**测试了 zenoh-pico 的字节处理、序列化与反序列化、复合数据类型支持、以及切片迭代器的功能，验证 zenoh-pico 在处理各种数据类型和结构时的正确性和稳定性。
2. **z\_get：**实现了一个zenoh pico客户端示例，主要用于演示如何通过zenoh协议进行查询操作。
3. **z\_get\_attachment：**实现了一个zenoh pico客户端示例，主要用于演示如何通过zenoh协议发送查询请求并处理带有附件和编码信息的回复
4. **z\_get\_channel：**实现了一个zenoh pico客户端示例，主要用于演示如何使用环形通道发送查询并接收处理回复
5. **z\_get\_liveliness：**该示例使用zenoh pico的活跃性查询功能检测网络中活跃的资源，同时验证了zenoh在多线程环境下的可靠性和动态配置的灵活性
6. **z\_info：**该示例使用 zenoh pico 框架获取网络中自身设备、路由器和对等节点的标识符（zid）
7. **z\_liveliness：**该示例验证了 zenoh pico 的 liveliness功能，具体实现了创建、声明和撤销生存性令牌的完整流程
8. **z\_ping：**该示例演示了 zenoh pico 的 ping-pong延迟测试，主要测试了发布和订阅的基础功能，以及消息传输的往返延迟和单向延迟
9. **z\_pong：**该示例实现了一个 zenoh pico pong响应器，主要用于处理来自 test/ping 路径的消息，并将其作为响应转发到 test/pong 路径
10. **z\_pub：**该示例实现了一个 zenoh pico 的发布者程序，功能包括动态配置、声明发布者以及循环发送数据到指定的路径表达式
11. **z\_pub\_attachment：**该示例实现了一个zenoh pico发布者程序，包含了消息的发布、附加元数据、编码、时间戳等方面的操作
12. **z\_pub\_st：**该示例实现了一个zenoh pico的发布者程序。其主要功能是通过zenoh发布数据，并且能够在特定的连接和监听模式下运行。代码中通过命令行参数配置一些基本的行为，循环发布数据，同时进行会话管理和心跳任务
13. **z\_pub\_thr：**该示例实现了一个基于zenoh pico的发布者程序，其主要功能是不断发布数据，并通过zenoh进行流式通信。包括命令行参数解析、配置初始化、会话打开、发布数据以及资源清理等步骤
14. **z\_pull：**测试了在 zenoh pico 中配置并启动一个订阅者，通过指定的 key expression 定期拉取数据，处理数据并输出 key-value 对。还包含了连接配置、任务启动、错误处理等基本操作。
15. **z\_put：**测试了通过zenoh pico库向指定的键表达式发送数据。用户可以通过命令行参数设置键、值、连接和监听器。程序打开会话、声明键表达式并执行数据发布，最后进行清理。
16. **z\_queryable：**测试了如何在 zenoh pico 中创建一个可查询的服务，通过定义查询回调处理函数，接收查询请求并根据配置返回数据、删除操作或错误响应。
17. **z\_queryable\_attachment：**测试了如何在 zenoh pico 中处理带有附件的查询请求，包括接收、解析和回复查询附带的键值对附件，同时演示了如何附加和序列化返回数据与附件。
18. **z\_queryable\_channel：**测试了如何使用 zenoh pico 创建一个支持查询的通道，并通过该通道接收查询请求，处理查询数据，生成响应并回复查询，同时演示了如何使用环形缓冲区处理查询消息。
19. **z\_scout：**测试了zenoh pico的侦测功能，展示如何使用 z\_scout来寻找网络中的Zenoh节点，并通过回调处理返回的Hello消息，打印节点的ID、类型和定位信息，同时管理回调过程中的资源。
20. **z\_sub：**测试了zenoh pico的订阅功能，演示了如何声明一个订阅者，接收指定主题的消息，并处理收到的数据，通过回调函数输出消息的键值对。同时，支持通过命令行参数控制订阅的主题和接收的消息数量。
21. **z\_sub\_attachment：**测试了zenoh pico订阅功能，展示了如何处理带有附件的消息。该示例接收带附件的数据，并使用反序列化技术解析附件内容，输出键值对信息以及消息的其他元数据，如时间戳和编码类型。
22. **z\_sub\_channel：**测试了zenoh pico的订阅功能，使用FIFO队列处理订阅消息。示例中，客户端订阅特定的主题表达式，并通过FIFO通道接收消息，打印接收到的键值对数据。
23. **z\_sub\_liveliness：**测试了zenoh pico的存活性（liveliness）订阅功能。客户端订阅特定主题，并根据收到的更新打印存活的令牌或被删除的令牌，用于检测数据源的存活性状态。
24. **z\_sub\_st：**测试了zenoh pico订阅功能，客户端订阅指定主题并接收消息。接收到的每条消息打印其键值对，直到达到指定的接收次数（默认为最大整数值）。
25. **z\_sub\_thr：**测试了zenoh pico订阅功能的高吞吐量性能，使用多线程订阅指定主题并统计接收到的消息数量。程序在接收到指定数量的消息后，计算并输出每秒接收的消息数量。

部分示例运行结果如下，示例运行全部通过。

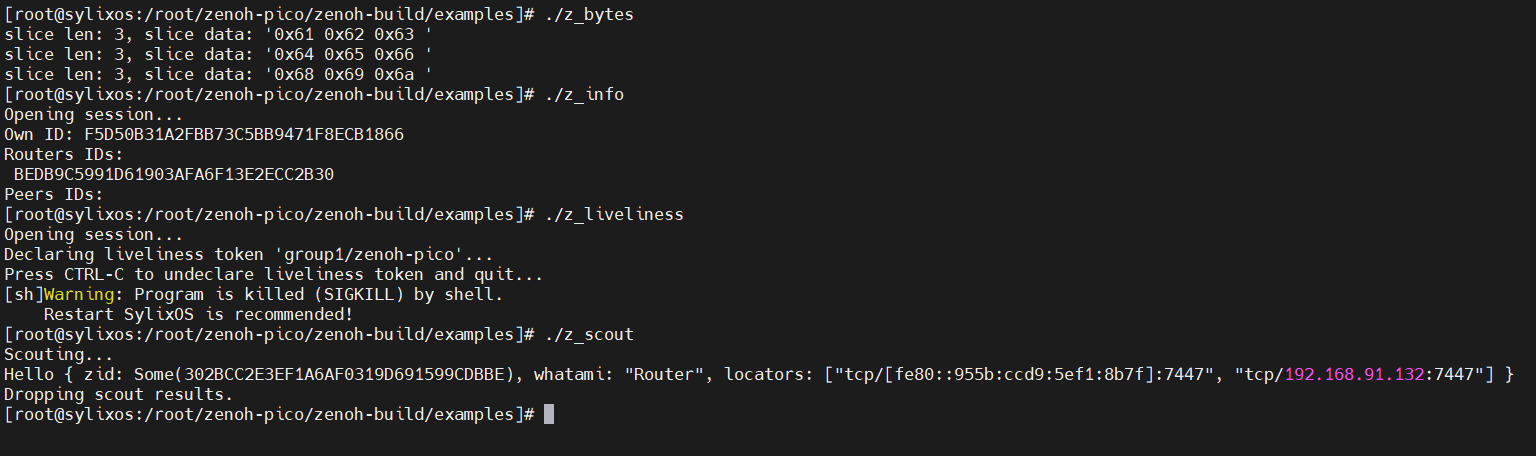


图31 部分示例运行结果

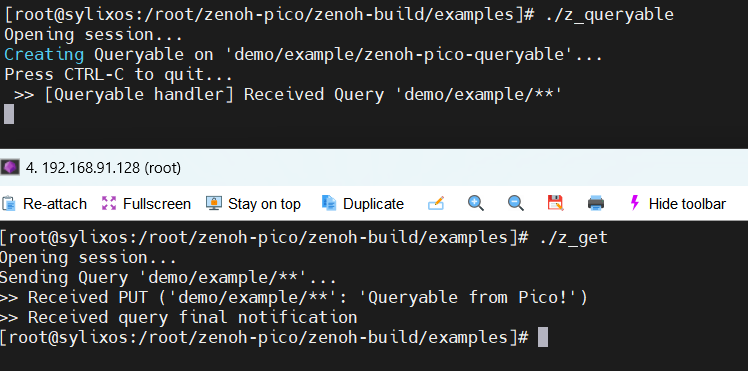


图32 部分示例运行结果

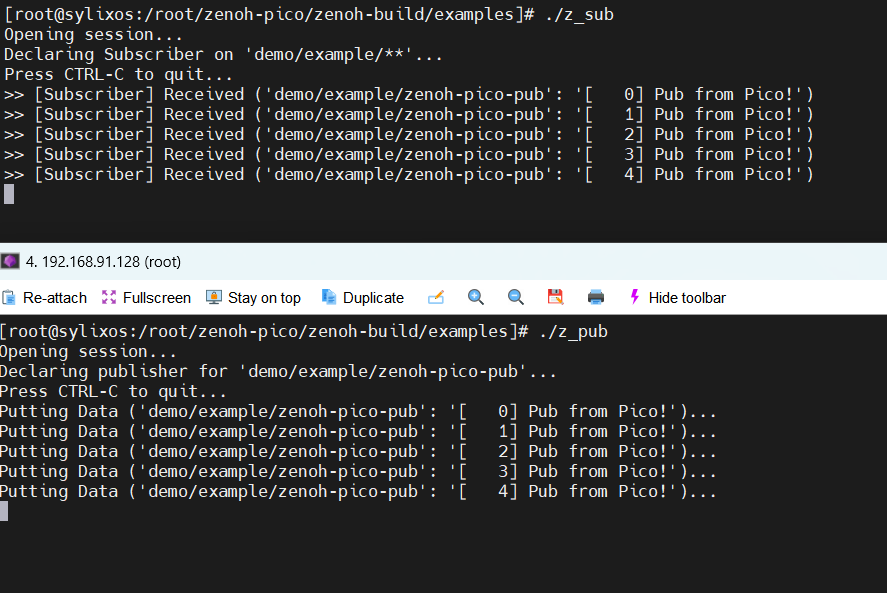


图33 部分示例运行结果

## 第六章 移植总结

在本次的移植工作中，我们成功地将Zenoh-Pico中间件移植到了SylixOS操作系统之上，并且通过了一系列的功能测试和性能验证。在移植的过程中，我们遇到了一些兼容性方面的问题，例如Zenoh-Pico不支持SylixOS平台、Zenoh-Pico和SylixOS内核函数命名重复等问题，经过我们几人的共同努力，这些问题都得到了有效的解决，从而确保了Zenoh-Pico能够在SylixOS的环境下运行。

通过这次移植工作，我们收获了许多。想要成功移植一个项目，先要理解项目内容和目标平台的信息，其次要掌握编译一个软件的具体流程，包括cmake、make等，要能够在编译出错时准确定位并修改错误的地方。此外，我们还深刻体会到团队合作的重要性。在遇到复杂的技术难题时，团队成员之间的有效沟通和协作是解决问题的关键。每个人都发挥了自己的专长，共同攻克了一个又一个难关。通过这次实践，我们的技术能力和项目经验都得到了显著提升，为今后类似项目的开展打下了坚实的基础。同时，我们也认识到文档编写和测试工作的重要性，完善的文档和严谨的测试是确保项目成功不可或缺的环节。

在这次两个项目的移植过程中，我们意识到自己对于知识的整合与连通仍需加强，此外，对于基础知识的掌握也不够透彻。首先，在TD-engine的移植过程中，我们没有在移植前深刻学习TD-engine的相关内容，没有预先确认这个项目的移植具体方案，直接进行交叉编译，这也是后面出现许多问题的关键。在make的时候，我们只能出现什么问题就先解决问题，不能对后面所出现什么问题作出预估，很多时候我们会被一个问题卡住，导致整个make的进程也就卡住。另一个原因就是TD-engine项目include了很多sylixos不支持的头文件，我们一直在寻找怎么解决头文件问题。所以在移植前我们应该先看一下相关程序，在确定具体的移植步骤，减少后面步骤的时间损失。在另一个项目zenoh-pico中，移植进程就稍通畅了些。所以，在移植的初期阶段，我们需要先对目标平台的硬件和软件环境进行深入了解，查阅相关的官方文档，学习不同平台之间的架构差异，了解底层系统的运行机制，这样有助于我们确定整体项目的移植步骤，在移植时也会更加顺利。除此以外，对于中间件移植、交叉编译等操作的具体知识内容也有所缺乏，只能在相关的文档和操作手册的支持下进行相关操作，应该在其余时间对其知识架构进行了解学习。除了理论学习与移植项目外，我们应该与老师保持沟通，遇到问题也需要进行讨论解决。总的来说，这是一场有意义的项目实习，从技术层面到团队合作，从问题解决到自我管理，都学到了许多实用的技能。