TEC | Tecnológico de Costa Rica

SETEC Lab
Space Systems Laboratory

Documento Tutorial

Manual de Uso de la Biblioteca VxMAES.

Brenda Valeria Lobo Mora

Cartago, 20 de febrero 2024

Resumen

En este manual se presenta la funcionalidad de la biblioteca VxMAES, basada en el paradigma MAES o Multi-Agent Systems Framework for Embedded System, VxMAES se encuentra en el contexto del sistema operativo en tiempo real VxWorks escrita en lenguaje de programación C. Se realizaron tres casos de uso para verificar el funcionamiento de VxMAES, generando proyectos del tipo Real Time Process (RTP) para cada caso: *Sender Receiver*, *Telephone Game* y *System Verification*,

Al utilizar esta guía, podrá generar la imagen del sistema operativo, junto con las aplicaciones, para integrarlas en el sistema embebido Raspberry Pi 4 - Modelo B, Se hace uso de la estación de trabajo Wind River Workbench, utilizado para adaptar, simular e incorporar el sistema operativo en un embebido. Cabe destacar que la biblioteca VxMAES no se limita a utilizarse exclusivamente para la Raspberry Pi 4 modelo B, sino que por medio de la plataforma Wind River Workbench se puedan explorar otros sistemas empotrados que la empresa Wind River disponga en sus paquetes de soporte.

Palabras clave: C, MAES, MAS, RTOS, RTP, Raspberry Pi 4, sistemas espaciales, VxWorks.

Abstract

This manual presents the functionality of the VxMAES library, based on the MAES paradigm or Multi-Agent Systems Framework for Embedded System, VxMAES is in the context of the VxWorks real-time operating system written in the C programming language. Three use cases to verify the operation of VxMAES, by generating projects of the Real Time Process (RTP) type for each application: *Sender Receiver*, *Telephone Game* and *System Verification*,

By using this guide, you will be able to generate the operating system image, along with the applications, to integrate them into the Raspberry Pi 4 - Model B embedded system. Wind River Workbench workstation is usted to adapt, simulate and incorporate the operating system in an embedded. The VxMAES library is not limited to being used exclusively for the Raspberry Pi 4 model B, but through the Wind River Workbench platform, other embedded systems that the Wind River company has in its support packages can be explored.

Keywords: C, CMAES, MAS, RTOS, RTP, Raspberry Pi 4,space systems, VxWorks.

Índice general

Capítulo 1

Componentes de la Biblioteca VxMAES

Antes de adentrarse en los elementos de VxMAES es importante considerar ciertos aspectos de sus versiones anteriores, MAES [1] y FreeMAES [2]. Estas bibliotecas, originalmente escritas en lenguaje C++, se basan en la programación orientada a objetos, en la cual se empleaban clases para cada componente del paradigma MAES. Sin embargo, CMAES está basado en el lenguaje de programación C, por lo que se implementó utilizando *structs* [3]. Este cambio implica la necesidad de vincular las funciones o métodos de cada clase a través de punteros. A pesar de esta transición, se ha procurado mantener la estructura en la que fue escrita dicha biblioteca, por lo que identifican los componentes como clases y se detallan sus respectivos atributos y métodos asociados.

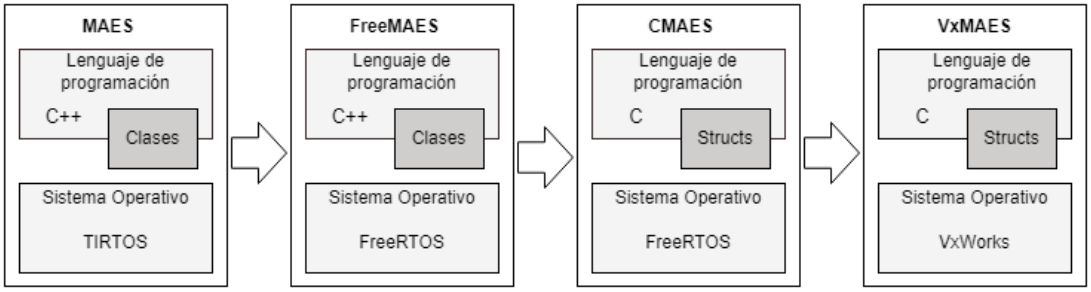

Figura 1.1: Historial de versiones biblioteca MAES.

1.1. Diagrama de Clases

Con el fin de mostrar la estructura general de la biblioteca, se representa en un diagrama de clases en la Figura 1.2. Cabe destacar que la tarjeta utilizada en este caso es una Raspberry Pi 4, que puede ser reemplazada por otro dispositivo que Wind River tenga soporte. De acuerdo con el diagrama, se tiene que el software a bordo de la Raspberry Pi 4 - modelo B, hace uso de VxWorks. Este sistema operativo contiene tanto su espacio kernel (donde siempre van a existir tareas del kernel en ejecución para la operación correcta del sistema completo) y su espacio de usuario o RTP (cero o un RTP en espacio de usuario debido a que esta biblioteca no tiene soporte para sistemas multiplataforma). En específico, para esta implementación el RTP va a usar la biblioteca desarrollada VxMAES, la misma posee un encabezado (*header* o archivo *.h*) de configuración.

El RTP persé, se mapea o extiende a la clase *Agent Platform*. La plataforma de agentes, puede tener una o más organizaciones (de la clase *Agent Organization*), se compone de una

relación uno a uno de condiciones de usuario (*User Definition Conditions*) y la clase *SysVars* de variables de ambiente. La clase *Agent* debe tener uno o más agentes, ya que son esenciales para que una organización o bien, plataforma, exista. Estos agentes deben tener sin excepción, ya sea un comportamiento cíclico o de una sola ejecución (clase *Cyclic Behavior* y *One Shot Behavior* respectivamente). Como resultado de los comportamientos y comunicación entre agentes, puede existir varias estructuras de mensaje de agentes de la clase *Agent Message* dentro de la plataforma. La clase *Agent Message* se extiende o mapea a colas de mensajes y la clase *Agent* se mapea a tareas en el SO de VxWorks.

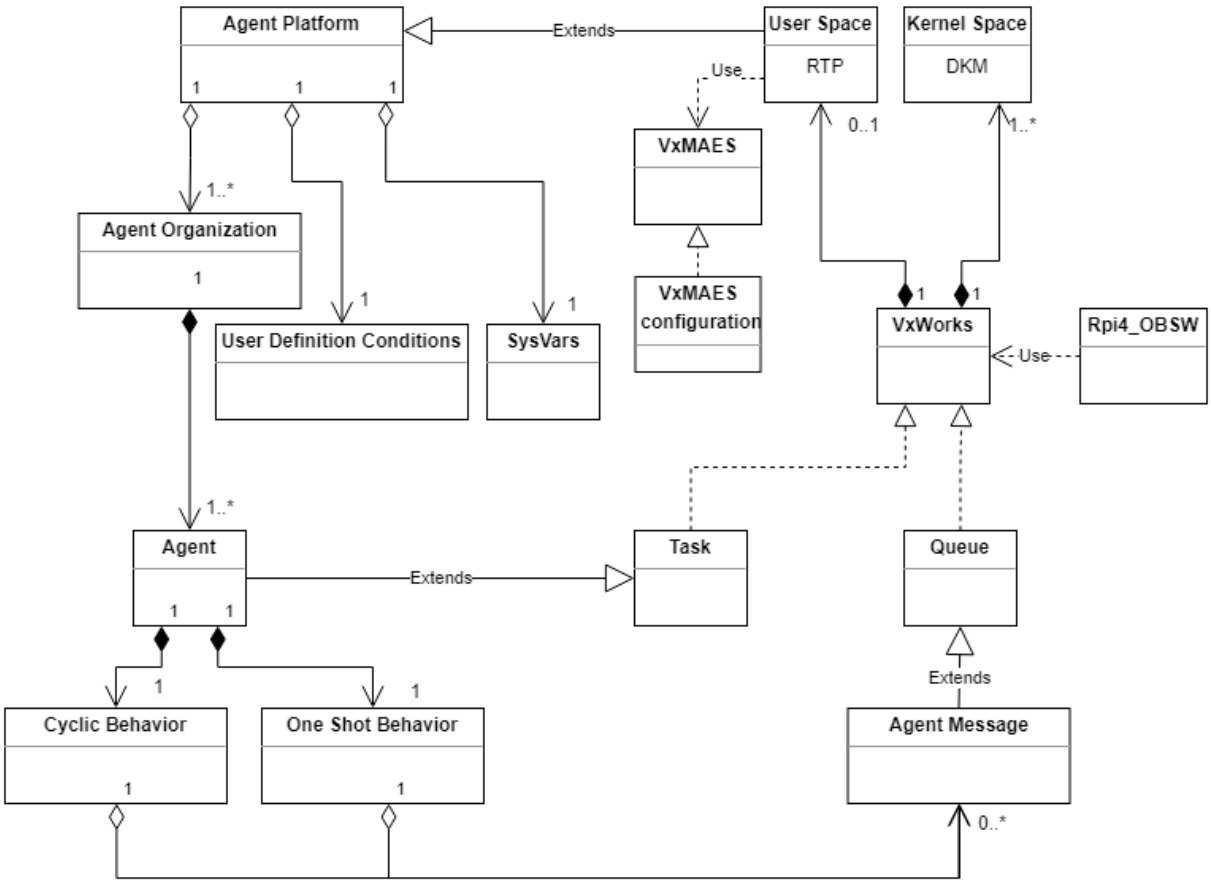

Figura 1.2: Diagrama de clases VxMAES.

1.2. Configuración de la Biblioteca

En una primera etapa, se establecen los conceptos clave de la biblioteca, creando así una conexión entre la biblioteca y el kernel del sistema operativo VxWorks. La configuración mencionada se agrega a un *header* llamado *VxMAESconfig.h*, para que el usuario tenga acceso rápido a los *define* y elementos que puede configurar de forma manual.

Primeramente, se incluyen aquellos *define* establecidas en VxWorks, los cuales guardan similitud con los *define* previamente establecidos en la biblioteca. Dentro de las principales se encuentra el *TASK_ID* que identifica cada tarea y el *MSG_Q_ID* que identifica la cola de mensajes. También, se define el tipo de dato que específica los argumentos o parámetros de las funciones *_Vx_usr_arg_t* (se agrega ya que en CMAES no se definía), así como los *ticks* del

sistema *_Vx_ticks_t*. Además, se establece un tipo de dato para las funciones que se vinculan a un agente tipo *void**, otro para datos cuyo tamaño se desconoce pero que por defecto, se asigna como tipo *long unsigned int* (debe ser sin signo, ya que se utiliza para representar números positivos), y por último, se define un tipo de datos para especificar el tamaño de una pila o *stack* como *size_t*.

```
/* MAES | VxWorks Definitions */

#define MAESTaskHandle_t TASK_ID            // Task Identifier
#define MAESQueueHandle_t MSG_Q_ID          // Message Queue
#define MAESArgument _Vx_usr_arg_t          // Parameter
#define MAESTickType_t _Vx_ticks_t          // Ticks type
#define MAESTaskFunction_t void*            // Function Pointer
#define MAESUBaseType_t long unsigned int   // integer
#define MAESStackSize size_t                // Stack type
```

Listing 1.1: Configuración de los tipos de datos.

Un segundo paso es nombrar los *define* propios de la biblioteca que identifican los agentes y las colas de mensajes.

```
/* MAES Definitions */

#define Agent_AID MAESTaskHandle_t // Agent ID
#define Queue_ID MAESQueueHandle_t // Agent's Queue ID
```

Listing 1.2: Identificador propio de VxMAES para agentes y colas de mensajes.

Además, se definen ciertos elementos que pueden ser modificados por el usuario, la cantidad de agentes, cantidad de comportamientos, cantidad de organizaciones propios de la implementación, ya que varía de acuerdo a la aplicación que se quiere desarrollar.

```

#define AGENT_LIST_SIZE 64              // Maximum Agents per
                                           platform
#define MAX_RECEIVERS AGENT_LIST_SIZE - 1 // Maximum receivers
                                          available for any agent
#define BEHAVIOUR_LIST_SIZE 8           // Behaviour list size
#define ORGANIZATIONS_SIZE 16           // Maximum Members by
                                           Organization
```

Listing 1.3: Configuración ajustable por el usuario (parte 1).

Como último paso, se define la prioridad máxima que se desea dar a la aplicación. En VxWorks la máxima prioridad son los números de menor valor, es recomendable que la prioridad esté dada entre 100 a 250, ya que los valores menores a 100 están destinados para otras tareas propias del kernel. Cabe destacar que el calendarizador de VxWorks basado en la prioridad toma en cuenta tanto tareas de kernel como de espacio de usuario, por esta razón es que hay números de prioridad que no se deben utilizar. En este caso se define la mayor prioridad, que va a ser dada al AMS de la aplicación como 100. También se define el tamaño mínimo del stack el cual se asigna igual a cómo se define en CMAES , y máxima longitud del mensaje

en bytes, este debe ser manipulado por cada desarrollador, considerando que si utiliza valores mayores le da un peso a la memoria.

```
#define MAESmaxPriority 100  // The Higher priority for application
#define MAESminStacksize 50  // Minimum stack size
#define MAXmsgLength 100     // Maximum Message Lenght (bytes)
#define ONE_MINUTE_IN_TICKS (sysClkRateGet() * 60) // 1 minute in ticks
```

Listing 1.4: Configuración ajustable por el usuario (parte 2).

1.3. Clase Agent

La clase *Agent* corresponde a la unidad agente, que es la base de los Sistemas Multi-Agentes. Esta clase se encarga de definir la estructura que compone a un agente, se establecen ciertos parámetros que posteriormente pueden ser cambiados por la clase *Agent Platform*. Esta clase se compone de los siguientes atributos y métodos:

- **Atributos**:
  - **Agent**: estructura que establece toda la información que contiene un agente en el paradigma. Dentro de la información se encuentra el AID (Agent IdentifiAer), el ID de la cola de mensajes, la plataforma a la que pertenece y la organización, los mismos inicializan en "NULL" dado que la clase *Agent Platform* define estos valores posteriormente. También contiene el nombre del agente, la prioridad y el tamaño del *stack*, los cuales son definidos inicialmente por el desarrollador en la aplicación, y el tipo de afiliación de la organización y el rol dentro de la organización, se establecen por defecto como *NON_MEMBER* y *VISITOR*, respectivamente.

```
   // Agent Information
typedef struct Agent_info Agent_info;
struct Agent_info
{
    Agent_AID aid;
    Queue_ID queue_id;
    char* agent_name;
    STATUS priority;
    Agent_AID AP;
    org_info* org;
    ORG_AFFILIATION affiliation;
    ORG_ROLE role;
};
```

Listing 1.5: Atributo agente en VxMAES.

  - **Resources**: estructura que almacena información para la tarea que realizará el agente, estos son: el tamaño del *stack* del agente, un puntero a la función a ejecutar, y los parámetros que se incorporan en la función.

```
    // Agent Resources
typedef struct Agent_resources
    Agent_resources;
struct Agent_resources
{
    MAESStackSize stackSize;
    MAESTaskFunction_t function;
    MAESArgument taskParameters;
};
```

Listing 1.6: Recursos reservados para un agente.

- **Métodos:**
  - AID: este método retorna el AID del agente.
  - Iniciador: este método establece los valores iniciales de los atributos del agente por defecto.

1.4. Clase SysVars

Esta clase *sysVars* consiste en las variables de ambiente de la biblioteca CMAES. Cabe destacar que VxWorks tiene soporte de variables de ambiente solamente para proyectos que se ejecuten en espacio kernel (DKM). Para proyectos de espacio de usuario (RTP), no existen variables de ambiente que realizan los mismos métodos establecidos en *SysVars* y es el tipo de proyecto que se desea trabajar. Por lo que es necesario utilizar las variables de ambiente ya definidas en la biblioteca, que fueron creadas dado que FreeRTOS no contaba con soporte para dichas variables.

Esta clase tiene la función de establecer los agentes dentro de un ambiente común, permitiendo tener un mayor control de lo que le pertenece a una aplicación basada en la biblioteca. Esta clase cuenta con un único atributo llamado *environment*, en el que se introduce el AID del agente y su respectivo puntero a la información. Dentro de los métodos se pueden nombrar:

- **get_taskEnv**: busca un agente del ambiente utilizando su AID, y devuelve un puntero a la estructura correspondiente al agente encontrado.
- **set_taskEnv**: registra un agente en un ambiente específico, se requiere proporcionar el ambiente designado, el AID del agente y su respectivo puntero.
- **erase_taskEnv**: elimina un agente que se encuentra registrado dentro del ambiente, se requiere proporcionar el AID del agente.
- **getEnv**: obtiene la estructura en donde se encuentran todos los elementos del ambiente.

1.5. Clase Agent Message

Esta clase permite la comunicación entre los agentes, en esta se definen las estructuras para los mensajes que se envían entre los agentes así como ciertos atributos y métodos que se explican a continuación.

- **Atributos**:
  - **Message**: estructura que contiene información de: el agente que envía el mensaje, el agente que recibe el mensaje, el tipo de mensaje (basado en el lenguaje de comunicación de agentes) y el contenido del mensaje.

```
// Message
typedef struct
{
    Agent_AID sender_agent;
    Agent_AID target_agent;
    MSG_TYPE type;
    char* content;
} MsgObj;
```

Listing 1.7: Estructura de contenido en mensaje.

  - **Ptr_env**: puntero al ambiente
  - **Receivers**: lista con los AID de los agentes que pueden recibir el mensaje.
  - **Subscribers**: contador que indica la cantidad de receptores del mensaje
  - **Caller**: brinda el AID del agente que enviará el mensaje.
- **Métodos**:
  - **Is Registered**: se encarga de verificar si el agente receptor y el emisor se encuentran en el mismo ambiente y plataforma, devuelve *true* o *false* dependiendo si se encuentra el AID del agente receptor.
  - **Get Mailbox**: retorna el ID de la cola de mensajes, al brindarle la instancia de la estructura de mensajes y el AID del agente.
  - **Agent Message**: se encarga de establecer en *default* los valores de algunos punteros. Se configura el ID del emisor del mensaje y limpia la lista de receptores.
  - **Add Receiver**: agrega a la lista de receptores el AID de un agente dado, a esta función se le debe proporcionar el AID del agente y este retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
  - **Remove Receiver**: borra el AID de un agente de la lista de receptores del mensaje, a esta función se le debe proporcionar el AID que se desea remover y este retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
  - **Clear all Receivers**: borra todos los agentes receptores que se encuentran en la lista y cambia sus valores por *NULL*.
  - **Refresh List**: se encarga de remover todos los AID de los agentes que se encuentran en la lista de receptores. La principal diferencia con la anterior es que no revisa la totalidad de la lista, sino que se basa en la cantidad de subscriptores que existen para remover dicha cantidad de agentes de la lista, de igual manera cambia sus valores a *NULL*.

- **Receive**: se encarga de recibir un mensaje de otro agente, toma como entrada la instancia de mensajes y los *ticks* de espera. Este último parámetro, si se coloca *WAIT_FOREVER*, la tarea se espera hasta recibir el mensaje y *NO_WAIT*, la tarea retorna de forma inmediata si no existe nada en la cola. Cuando el mensaje se recibe correctamente, la función retorna la cantidad de bytes que fueron copiados en el *buffer*.
- **SendX**: Esta función se encarga de enviar un mensaje a otro agente, tiene como entradas la estructura de mensaje, el AID de un agente receptor, y los *ticks* de espera. Este último parámetro, si se coloca *WAIT_FOREVER*, la tarea se espera hasta enviar el mensaje. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Send All**: método encargado de enviar el mensaje a todos los agentes que se encuentren en la lista de receptores. Como parámetro, recibe la instancia de estructura de mensajes y retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Set msg Type**: permite establecer el tipo de mensaje, al colocar como entrada un valor numérico en el rango de 0 a 21, que está asociado a un tipo de mensaje definido.
- **Set msg Content**: coloca el contenido del mensaje que se desea transmitir.
- **Get Msg**: retorna el puntero a la estructura del mensaje.
- **Get msg Type**: obtiene el tipo de mensaje.
- **Get msg Content**: se encarga de obtener el contenido de un mensaje.
- **Get Sender**: obtiene el AID del agente emisor del mensaje.
- **Get Target**: obtiene el AID del agente receptor del mensaje.
- **Registration**: envía una solicitud al AMS para poder registrar un agente en la plataforma de agentes, se debe ingresar el AID del agente como entrada. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Desregistration**: envía una solicitud al AMS con el fin de borrar un agente del registro de la plataforma, se debe ingresar el AID del agente como entrada. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Suspend**: envía una solicitud al AMS con el fin de suspender un agente dentro de la plataforma, se debe ingresar el AID del agente como entrada. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Resume**: envía una solicitud al AMS con el fin de reanudar la ejecución de un agente dentro de la plataforma, se debe ingresar el AID del agente como entrada. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Kill**: envía una solicitud al AMS con el fin de eliminar un agente dentro de la plataforma, se debe ingresar el AID del agente como entrada. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.

- **Restart**: envía una solicitud al AMS con el fin de restablecer un agente dentro de la plataforma, se debe ingresar el AID del agente como entrada. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.

1.6. Clase Agent Platform

Esta clase hace referencia a la plataforma de agentes, cuyo contenido puede ser compuesto por varias organizaciones, que a su vez estas organizaciones se componen de uno o más agentes. En general, la plataforma representa una estructura física para los agentes. Cabe destacar que, el paradigma MAES establece que solo exista una plataforma por hardware o sistema embebido.

La clase *Agent Platform* incluye la funcionalidad del AMS para regular los agentes registrados en la plataforma y los servicios que ofrecen. La clase se compone por los siguientes atributos y métodos:

- **Atributos:**
  - **Ptr_env**: corresponde a un puntero hacia el ambiente.
  - **AMS Agent**: corresponde a la estructura o información del sistema Administrador de agentes.
  - **Agent Handle**: lista que contiene los AID de los agentes que se encuentran dentro de la plataforma.
  - **AP Description**: corresponde a una estructura que contiene información de la plataforma como: el AID del AMS, los agentes subscritos en ella y el nombre de la plataforma. Por otra parte, en VxMAES se agrega otro elemento para almacenar el ID del RTP. El RTP inicializa la ejecución de las tareas contenidas dentro de este, por lo que el ID del RTP se utiliza en los diferentes métodos de la plataforma. El RTP siempre va a contener una prioridad mayor a las tareas hijas.

```
// AP Description
typedef struct
{
        Agent_AID AMS_AID;
        Agent_AID RTP_info;
        char* AP_name;
        MAESUBaseType_t subscribers;
} AP_Description;
```

Listing 1.8: Descripción de la Plataforma de Agentes.

  - **Condition**: condiciones establecidas para la plataforma donde se indica las limitaciones y aceptaciones del AMS.
  - **Ptr_cond**: puntero a las condiciones de la plataforma (clase *User Defined Conditions*).
  - **AMS Parameter**: parámetros que alberga el AMS del sistema completo, contiene una instancia de la estructura de las variables de ambiente, una instancia del

tipo plataforma (corresponden a los atributos de la clase *Agent Platform*), y una instancia de las condiciones de usuario.

```
/*  Class: AMS Task */
struct AMSparameter{
        sysVars* ptr_env;
        Agent_Platform* services;
        USER_DEF_COND* cond;
};
```

Listing 1.9: Estructura de los parámetros del AMS.

- **Métodos**:
  - **AMS Task**: esta función pertenece al AMS de la plataforma, por lo cual debe tener la mayor prioridad y siempre está en espera de un mensaje del tipo "REQUEST", que lo envían los agentes que quieren solicitar un permiso. Estos permisos se refieren a acciones como registrar, borrar del registro, suspender, reanudar, eliminar y restaurar un agente.
  - **Agent Platform**: establece los parámetros iniciales de la plataforma, entre ellos: parámetros, nombre del agente AMS, nombre de la AP, ID del RTP, cantidad de suscriptores (por defecto es 0), puntero a las condiciones de la plataforma (por defecto) y la prioridad del AMS.
  - **Agent Platform with Conditions**: establece los parámetros iniciales de la plataforma entre ellos: parámetros, nombre del agente AMS, nombre de la AP, ID del RTP, cantidad de suscriptores (por defecto es 0), puntero a las condiciones de la plataforma (definido por el usuario) y la prioridad del AMS.
  - **Boot**: realiza el arranque de la plataforma. Este método crea e inicializa el AMS y registra cada agente dentro de la plataforma con el método *Register Agent*, seguidamente los activa.
  - **Agent Init**: este método se encarga de crear un buzón de mensajes para el agente, crea un agente y lo vincula con su comportamiento y finalmente registra al agente dentro del ambiente.
  - **Agent Init with Parameters**: la diferencia con el método anterior es que inicializa un agente con los parámetros que define el desarrollador.
  - **Agent Search**: determina si un agente se encuentra dentro de la lista de agentes que pertenecen a la plataforma o no.
  - **Agent Wait**: genera un atraso en la ejecución de la tarea en la plataforma al brindar una cantidad de ticks.
  - **Agent Yield**: utilizado para generar un cambio de contexto y ejecutar la tarea que llama esta función.
  - **Get Running Agent**: verifica el agente en ejecución y devuelve el AID del agente.
  - **Get State**: retorna el estado en que se encuentra el agente en base al AID del agente.
  - **Get Agent**: obtiene la información o descripción de un agente, al proporcionar un AID del agente como entrada.

- **Get AP Description**: obtiene la información de una plataforma, al proporcionar como entrada su AID.
- **Register Agent**: registra un agente dentro de la plataforma y lo activa si la lista de agentes no está llena. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Deregister Agent**: se encarga de borrar del registro a un agente de la plataforma, se da el AID del agente como entrada. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Suspend Agent**: suspende un agente dentro de la plataforma, al proporcionarse el AID del agente. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Kill Agent**: elimina un agente dentro de la plataforma, para ello primero lo debe borrar de la lista de agentes registrados, eliminar la cola de mensajes y eliminar la tarea.
- **Resume Agent**: reanuda la ejecución de un agente dentro de la plataforma, al brindarle el AID del agente como entrada. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Restart Agent**: restablece la ejecución de una gente en la plataforma, al proporcionarle el AID del agente como entrada.

1.7. User Defenition Conditions

Esta clase se encarga de determinar lo que quiere el usuario para su aplicación y si desea brindarle al AMS los permisos para realizar sus acciones y administrar los agentes. Los métodos se comentan adelante:

- **Métodos**:
  - **Register Condition**: habilita o deshabilita la posibilidad del AMS de registrar agentes.
  - **Kill Condition**: habilita o deshabilita la posibilidad del AMS de eliminar agentes.
  - **Deregister Condition**: habilita o deshabilita la posibilidad del AMS de borrar del registro a un agente.
  - **Suspend Condition**: habilita o deshabilita la posibilidad del AMS de suspender un agente.
  - **Resume Condition**: habilita o deshabilita la posibilidad del AMS de reanudar un agente.
  - **Restart Condition**: habilita o deshabilita la posibilidad del AMS de restablecer un agente.

1.8. Clase Cyclic Behaviour y One Shot Behaviour

Estas clases se encargan de ejecutar el comportamiento de un agente, se dividen en cuatro secciones, la primera es establecer las configuraciones previas del comportamiento (por lo que solo se configura una vez). Seguidamente, la sección *action*, es el comportamiento que se va a realizar una vez o de forma cíclica. Por otro lado, si es necesario, se incorpora la sección de corrección y detección de errores. Que esta debe ser definida por el creador de la aplicación, las condiciones y qué métodos aplicar. Finalmente, en el espacio de *done* se define como *"true"* si es un comportamiento de una sola ejecución, o *"false"*, si es un comportamiento cíclico. En la Figura 1.3, se muestra el flujo que lleva a cabo un comportamiento.

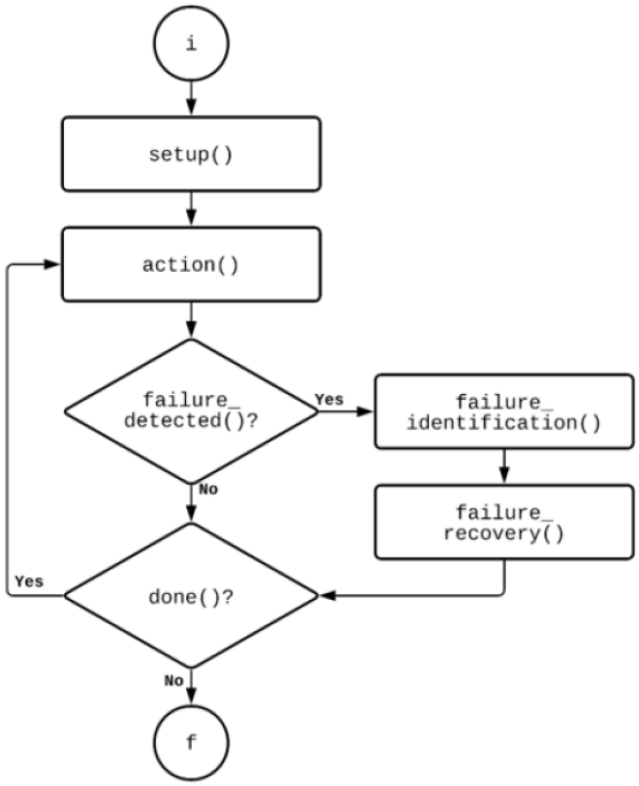

Figura 1.3: Diagrama de flujo comportamiento [2].

Los métodos y atributos se mencionan a continuación:

- **Atributos**:
  - **TaskParameters**: parámetros que permiten la ejecución de la función vinculada al agente.
  - **Message**: estructura del mensaje del agente.
- **Métodos:**
  - **Action**: contenedor del comportamiento del agente.
  - **Setup**: contenedor de la configuración previa del comportamiento del agente, se debe llamar antes del método *action*.

- **Done**: determina si la acción se ejecuta una o de forma cíclica.
- **Failure Detection**: contenedor de algoritmo para detección de fallas.
- **Failure Identification**: contenedor de algoritmo para desarrollar un identificador de fallas.
- **Failure Recovery**: contenedor de algoritmo para desarrollar una recuperación de la aplicación en caso de fallas.
- **Execute**: este es un contenedor que se encarga de seguir el ciclo de ejecución de los comportamientos del agente.

1.9. Clase Agent Organization

Esta clase se encarga de definir las funciones de una organización de los agentes, estas administran los miembros y asignan roles a los agentes miembros de una plataforma. El objetivo de esta clase es limitar la comunicación entre agentes, por lo que se dan en dos tipos: equipo o jerarquía. El tipo de equipo, permite la comunicación libre entre agentes dentro de una organización, pero restringida al asignar un miembro para que pueda comunicarse con agentes fuera de dicha organización. El tipo jerarquía, establece jerarquías dentro de una organización, por lo que los agentes solo pueden comunicarse con un *supervisor*.

- **Atributos**:
  - **ptr env**: puntero al ambiente.
  - **description**: descripción general de la organización.
- **Métodos**:
  - **Agent Organization**: se encarga de configurar los parámetros iniciales de la organización de agentes, se debe brindarle la instancia de la organización y el tipo.
  - **Create**: se encarga de crear una organización con parámetros iniciales, al brindar la instancia de la plataforma y de la organización. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
  - **Destroy**: elimina una organización de la plataforma al brindar la instancia de la organización. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
  - **Is member**: busca un agente dentro de la organización al brindarle la instancia de la organización y el AID del agente. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
  - **Is Banned**: determina si un agente ha sido expulsado de la organización, recibe la instancia de la organización y el AID de un agente. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
  - **Change Owner**: permite cambiar a dueño de organización a un agente mediante su AID y la instancia de la organización. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.

- **Set Admin**: se encarga de colocar el AID brindado como administrador de la organización. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Set Moderator**: se encarga de definir a un agente dado como el moderador de la organización, recibe la instancia de la organización junto con el AID del agente. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Add Agent**: se encarga de agregar un agente dentro de la lista de agentes de una organización, al brindarle la instancia de la organización y el AID del agente. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Kick Agent**: expulsa un agente de una organización al brindarle el AID del agente y la instancia de la organización. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Ban Agent**: se encarga de *banear* (restringir) un agente de la organización al brindarle el AID del agente. Si se encuentra entre los miembros se utiliza la función *kick* para *banearlo*. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Remove Ban**: remueve el estado de *baneo* de un agente dando el AID del agente y la instancia de la organización. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Clear Ban List**: limpia la totalidad de una lista de agentes que se encuentren *baneados* de la organización. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Set Participant**: configura un agente con el rol de participante al brindarle la instancia de la organización y el AID del agente. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Set Visitor**: configura el rol de un agente como *visitor* de la organización, al brindarle la instancia de la organización y el AID del agente. Este método retorna un valor numérico en el rango 0 a 7, el cual posteriormente es asociado a un error definido en la biblioteca.
- **Get Organization Type**: obtiene el tipo de organización al brindarle la instancia de la organización. Devuelve el tipo de organización.
- **Get Information**: obtiene la información de una organización al brindar la instancia de la organización.
- **Get size**: obtiene la cantidad de los miembros de una organización al brindar la instancia de la organización.
- **Invite**: invita a un agente a ser agregado a la organización y si acepta es agregado con función *add_agent*, devuelve un valor numérico en el rango de 0 a 21, que está asociado a un tipo de mensaje definido.

Capítulo 2

Uso de VxMAES

Este capítulo muestra el proceso que se lleva a cabo para generar aplicaciones que utilicen la biblioteca VxMAES. Se desarrollan al menos tres aplicaciones con el propósito de mostrar que la comunicación entre agentes y que el ciclo de vida de los agentes funcione correctamente. Por consiguiente, se realizan simulaciones utilizando esta biblioteca en el Workbench de Wind River

2.1. Plataforma de Desarrollo

El entorno de desarrollo integrado (IDE) que se utiliza para la biblioteca VxMAES es Wind River Workbench (versión 4.15.0), basado en Eclipse. La licencia universitaria proporcionada por Wind River al Tecnológico de Costa Rica, contiene la versión del sistema operativo VxWorks 21.07. Cabe destacar que versiones anteriores o posteriores ofrecen beneficios distintos, por o que se debe considerar al desarrollar este tipo de proyectos. La documentación respectiva se puede consultar en información en línea: **Versiones VxWorks**, esta requiere la solicitud de un usuario para acceder a toda la información, o bien, se puede encontrar los documentos dentro de la carpeta de instalación en la máquina *host*: *InstallDir/vxworks/21.07/docs*.

Para incluir la biblioteca se utiliza el proyecto del tipo *Shared Library* del Workbench, que permite generar bibliotecas y vincularlas dinámicamente a proyectos de desarrollo, en este caso se utilizan RTP (Real Time Process), para proteger las aplicaciones generadas, beneficiando que se trabaja en espacio de usuario. Por otro lado, el depurador (*debugger*) del Workbench provee de forma visual las tareas del sistema que se llevan a cabo en tiempo real, ya sea que se encuentren en ejecución, suspendidas o pendientes, por lo que facilita las pruebas realizadas a la biblioteca. Además, también incluye una herramienta llamada *System Analyzer* que ofrece funciones avanzadas de análisis para observar la secuencia temporal de la ejecución de aplicaciones.

2.2. Aplicaciones Desarrolladas con VxMAES

Se validan tres casos de uso que utilicen la biblioteca VxMAES, para observar el comportamiento de estas en VxWorks, las aplicaciones de prueba que se van a utilizar son: *sender_receiver*, *telemetry* y *telephone_game*. Antes de explicar el funcionamiento de estas aplicaciones, se detalla una estructura básica que debe poseer una aplicación.

En primer lugar, se deben incluir las bibliotecas necesarias, en este caso siempre se requiere de *VxMAES.h* y *vxWorks.h*, de acuerdo a los requisitos de cada aplicación se añaden bibliotecas adicionales. En segundo lugar, se incluyen los *enums* o estructuras, relacionadas directamente con condiciones o definiciones que el desarrollador desee incluir como por ejemplo parámetros en las funciones de cada agente. Como tercer lugar, se definen las variables de la aplicación, que corresponden a estructuras propias de la biblioteca. En cuarto lugar, se establecen las funciones y tipos de comportamiento que va a poseer cada agente, estos deben ser programados por el desarrollador de acuerdo al objetivo de la aplicación. Es posible que un tipo de comportamiento pueda ser usado por 1 o más agentes. Como último lugar, se establece el *main*, que define el punto de inicio de ejecución de cada aplicación RTP.

Dentro del *main* se inicializa un contador de *ticks* hasta cumplir con un tiempo de ejecución específico. También aquí se define el constructor de clases (refiriéndose por clases a las estructuras en C), el cual es esencial, ya que este permite construir todas las estructuras que pertenecen a la aplicación completa. Se continúa por la obtención del ID del RTP, que es característica de esta versión de la biblioteca, ya que se trabaja con proyectos RTP. Seguidamente, se instancia el iniciador de agentes y de plataforma, que establece los valores por defecto de cada uno. Y finalmente, el sistema de arranque que inicializa el sistema completo, que corresponde a los agentes que se crean dentro de la aplicación. En el Listado 2.1, se muestra la estructura del código de cada aplicación RTP.

```
/* Aplicacion basada en VxMAES*/

#Seccion de includes

#Seccion de enums y structs

#Seccion de definiciones o variables globales

#Seccion de comportamiento
    /*Codigo relacinado a los metodos y funciones
    para realizar la tarea propuesta */

    :: Setup
    :: Action
    :: Done
    :: Execute

#Seccion del main
    /* Codigo relacionado con la declaracion de structs
    de las instancias y sus respectivas asociaciones*/

    :: Constructores
    :: ID RTP
    :: Inicializadores
    :: Sistema de arranque
    :: Bucle tiempo de ejecucion
```

Listing 2.1: Estructura general del código de las aplicaciones por probar.

Cabe mencionar, que la plataforma de agentes está integrada dentro de una aplicación del tipo RTP, la cual debe incluir siempre el sistema de transporte de mensajes (MTS) y el Agente AMS. Al tratarse de un entorno RTP, toda la ejecución tiene lugar en un espacio de memoria independiente al del kernel, lo que garantiza la protección del sistema operativo.

2.2.1. Resultados esperados de las aplicaciones

A continuación, se brinda una pequeña descripción de las aplicaciones que contiene la biblioteca como ejemplo (*sender_receiver*, *telemetry* y *telephone_game*) y lo que se debe visualizar en la terminal al ejecutarlas.

2.2.1.1. Sender Receiver

Se opta por utilizar esta aplicación debido a su capacidad para representar de forma sencilla la comunicación entre dos agentes: un *sender* o emisor y un *receiver* o receptor. Esta aplicación consta de dos comportamientos específicos para cada agente, el comportamiento de *escribir* para el *sender* y el comportamiento de *leer*, para el *receiver*. Ambos comportamientos se realizan de forma cíclica de la clase *Cyclic Behaviour*.

```
------Sender Receiver APP------
 VxMAES booted successfully
        Initiating APP

***Sending Message***
***Receiving Message***
Message content: This is the message.

***Sending Message***
***Receiving Message***
Message content: This is the message.

****** VxMAES app execution stops *******
```

Figura 2.1: Salida en la terminal del *Wind River Simulator* al ejecutar la aplicación *Sender-Receiver*.

2.2.1.2. Telephone Game

Se ha desarrollado esta aplicación con el propósito de ilustrar el uso de un comportamiento de ejecución única llamado *One Shot Behaviour*. La aplicación consta de tres agentes: *Person 1*, *Person 2* y *Person 3*. El escenario planteado implica que estos tres agentes contienen un mensaje personal específico cada uno, y deben compartirlo siguiendo el orden de 1 a 3. Se debe agregar al final del mensaje recibido, el mensaje propio de cada agente, hasta formar la oración *"This is the message"*. Esta aplicación simula el juego que se conoce coloquialmente como "Teléfono Roto". El objetivo fundamental de la aplicación verificar si el mensaje transmitido entre los tres agentes llega correctamente al final, de forma tal que el mensaje del agente *Person 3* muestre el mensaje que ha sido enviado por todos.

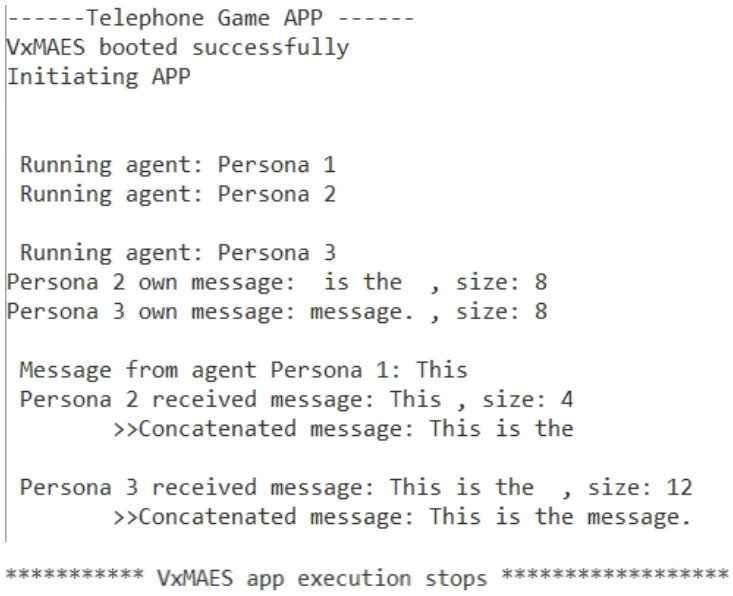

Figura 2.2: Salida en la terminal del Wind River Simulator al ejecutar la aplicación *Telephone Game*.

2.2.1.3. System Verification

Muestra el comportamiento de ejecución cíclica llamado *Cyclic Behaviour* de cuatro agentes: *Water temperature*, *Oil pressure*, *Tire pressure* y *Inspector*. El escenario planteado consiste simular la obtención de datos por sensores de presión y temperatura de un sistema de transporte, para indicar si es recomendable arrancarlo e indica si las mediciones se mantienen entre los límites. Cada agente de medición, brinda un valor de medición y se lo envía al agente *inspector*, este mantiene un orden de ejecución de los agentes. También, si se trabaja múltiples núcleos en un procesador, es importante mantener un tiempo de espera o *rate*, para que no interfiera la ejecución de las tareas cuando envían la medición al agente inspector, ya que se podrían tener resultados no esperados. Por esta razón, antes de enviar la medición al *inspector*, el agente debe esperar un periodo definido.

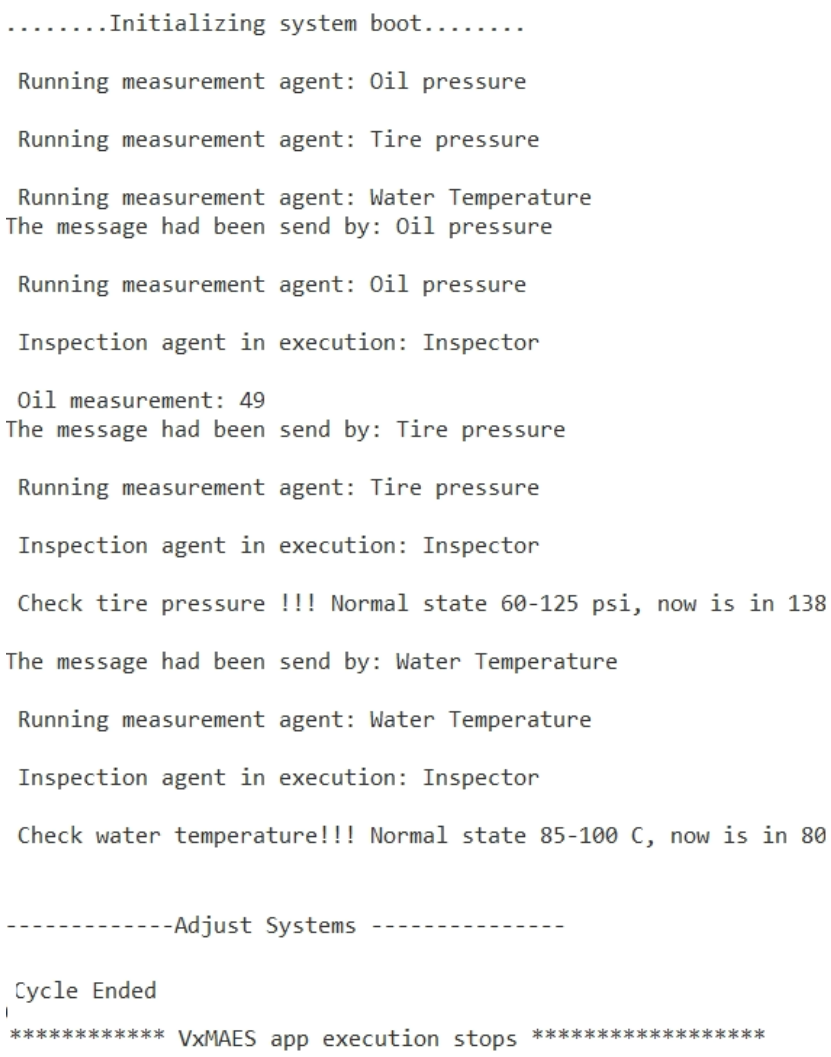

Figura 2.3: Salida en la terminal del Wind River Simulator al ejecutar la aplicación *System Verification*.

Capítulo 3

Integración de las Aplicaciones

En este capítulo, se da una breve descripción de la arquitectura del sistema y los proyectos que se deben generar tanto del sistema operativo como de las aplicaciones. Seguidamente, se muestra el proceso para integrar las aplicaciones desarrolladas ya sea en el *Wind River Simulator* o en el sistema embebido Raspberry Pi 4.

3.1. Arquitectura de Aplicaciones en VxWorks

En VxWorks se siguen una serie de pasos para obtener el ejecutable del producto final, que se describen a continuación:

1. **Proyectos de Sistema Operativos:**

   a) Se configura y construye VxWorks mediante la generación del proyecto VxWorks Source Build (**VSB**), basado en la CPU, Board Support Package (**BSP**) de la tarjeta destino o un BSP generado por un tercero. Se agregan las *layers* (capas) necesarias (controladores, bibliotecas de Kernel, BSP, *middleware* del sistema operativo, herramientas de depuración).

   b) Crear la VxWorks Image Project (**VIP**), que corresponde a la imagen del sistema operativo, en ella se deben seleccionar los componentes y bibliotecas del VSB que se desean incorporar a la imagen. También compila los archivos fuente del BSP y vincula a los componentes seleccionados anteriormente.

2. **Proyectos de Aplicaciones**

   a) Se crean aplicaciones **RTP** en espacio de usuario, en el que se genera un binario enlazado a un ejecutable de extensión *.exe*.

   b) Se integra la aplicación a la imagen de VxWorks por medio del sistema de archivos *ROMFS* en el que se agrega el ejecutable. Cabe mencionar, que se puede agregar cualquier tipo de archivo, en este caso se agrega también las bibliotecas necesarias (*VxMAESbib.so, libc.so.1 y libllvm.so.1*). Al generar la imagen se coloca en una memoria *flash* dentro del embebido.

   c) Se realiza el arranque de la tarjeta, ya sea simulada (estación de trabajo en máquina *host*) o en una tarjeta física. Se carga y se depura la aplicación generada por medio de la línea de comandos.

Estos pasos pueden representarse de manera gráfica mediante un diagrama de arquitectura del sistema (ver Figura 3.1), el cual consta de tres capas: la capa del desarrollador, la capa de software y la capa de visualización. En la capa del desarrollador, se configuran los proyectos en el espacio de trabajo *host* con el Workbench, definiendo los componentes específicos para las aplicaciones y la tarjeta física deseada. La capa de software comprende los archivos generados a partir de los proyectos, como el ejecutable del RTP y la imagen del sistema operativo. Por último, la capa de hardware está formada por el embebido Raspberry Pi 4 y memoria flash (tarjeta SD), donde se almacenan los archivos esenciales para la inicialización de la imagen del sistema operativo.

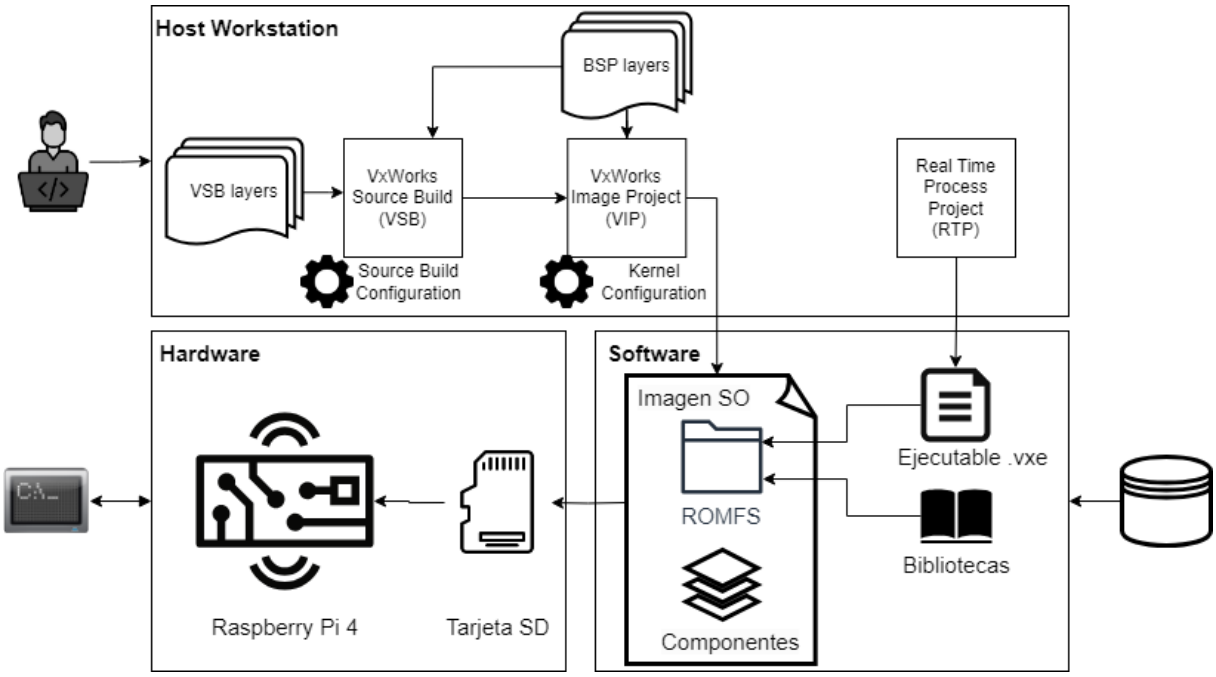

Figura 3.1: Arquitectura del sistema.

3.2. Tutorial para Simular los Casos de Uso en el Workbench.

Para la simulación de los casos de uso, se requiere: crear un proyecto VxWorks Source Build (VSB), crear un proyecto VxWorks Image Project (VIP) basado en el VSB generado, crear un proyecto Shared Library Project que contenga la biblioteca VxMAES y finalmente generar un proyecto Real Time Process (RTP) que contenga la aplicación de caso de uso. Para la sección de generar el VSB y VIP, se sigue el capítulo 2 y capítulo 3 de la guía "VxWorks Real Time Process Application Tutorial- Workbench 4 - 21.07". Esta guía se encuentra en la dirección de instalación del Workbench: */WindRiver/vxworks/21.07/docs/GettingStarted/*.

Seguidamente, se indica el proceso para generar una biblioteca compartida y la manera de incorporar VxMAES en un proyecto RTP.

3.2.1. Generar Biblioteca Compartida con VxMAES.

1. Copiar el contenido de la biblioteca VxMAES en la sección de documentos de su computadora host. Descargar el archivo *zip* del directorio de git: VxMAES.

2. En el Workbench generar un nuevo proyecto en: File → New → Wind River Workbench Project.

3. En la ventana de *Build type*, seleccionar *Shared User Library*. Seguidamente, se selecciona "next".

4. Se coloca el nombre de la biblioteca y se selecciona "next".

5. Se asegura que en la casilla "project" esté seleccionado el VSB correspondiente al proyecto.

6. Se da clic derecho al proyecto de la biblioteca creada y se selecciona: import → General → File System. Seguidamente presione "next" y busque el directorio donde descargó la biblioteca VxMAES. Presione "Select All" para seleccionar todos los archivos fuente y headers de la biblioteca y finalmente presione "Finish".

7. Se da clic derecho al proyecto de la biblioteca creada y se selecciona: → build →. Si se despliega una ventana de *build*, seleccione el botón "generate includes". El contenido de la biblioteca debe coincidir con la Figura 3.2.

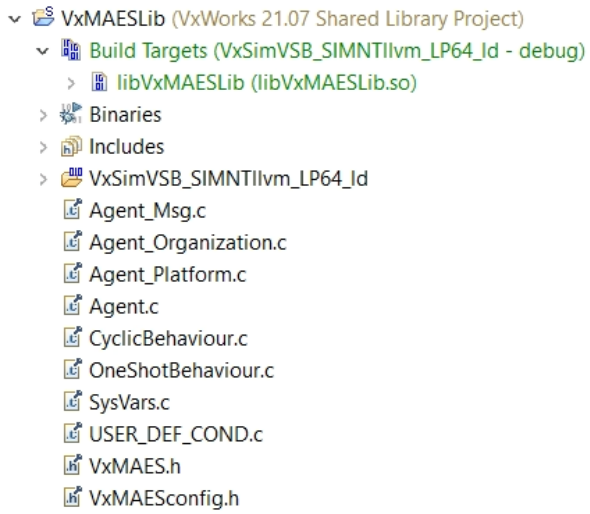

Figura 3.2: Contenido de la biblioteca VxMAES.

8. Al desplegarse la ventana de *analyze include directives*, mantener la configuración por defecto (está desmarcada la opción "Ignore all system include directives"). En la siguiente ventana seleccione "finish".

9. En la ventana emergente llamada "C/C++ Index configuration changed" seleccione "yes".

3.2.1.1. Creación de Proyectos Real Time Process que Utilicen VxMAES

Para generar proyectos RTP que se basen en una biblioteca, es necesario seguir la guía del capítulo 8 llamado "Example Implementation of a Shared Library" de la guía "VxWorks Kernel Application and RTP Application Projects Guide - 21.07". Esta guía se encuentra en la dirección de instalación del Workbench: */WindRiver/vxworks/21.07/docs/Core/*.

En este mismo capítulo se deben seguir los siguientes tutoriales:

- *Creating the RTP Application*: este es un tutorial para generar una aplicación RTP, se encuentra en la página 40. Para la aplicación RTP se pueden tomar los tres casos de uso *Sender Receiver*, *Telephone Game*, *Verification System* que se encuentran en el archivo de descarga de la biblioteca. Para importar un RTP se selecciona: import → General → File System. Seguidamente presione "next" y busque el directorio donde descargó la biblioteca VxMAES y seleccione un archivo fuente que corresponda al RTP deseado. Seguidamente seleccione "Finish".

- *Setting up a Build Target for fooRtpApp*: este tutorial permite vincular la aplicación RTP a la biblioteca compartida previamente generada, se encuentra en la página 42. Al editar el contenido de cada RTP de las aplicaciones, debe coincidir con la Figura 3.3.

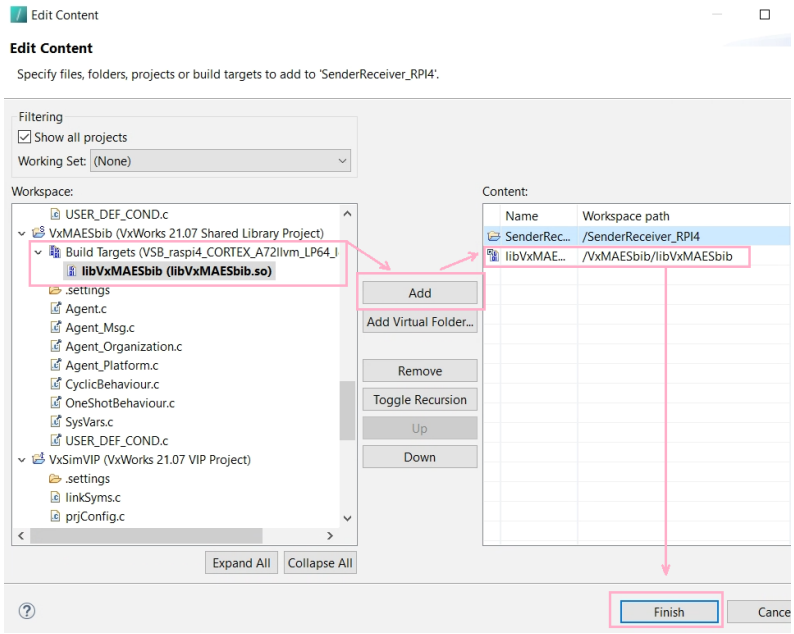

Figura 3.3: Sección de *edit content* de cada RTP para vincular la biblioteca VxMAES.

- *Connecting to the Target and Running the Example RTP Application*: este tutorial se utiliza para ejecutar proyectos RTP que se basen en una biblioteca, además muestra la forma de vincular las bibliotecas al *VxWorks Simulator*. Se puede encontrar en la página 48.

3.3. Tutorial para Utilizar los Casos de Uso en la Raspberry Pi 4 - Modelo B

3.3.1. Configuración Previa para la Tarjeta SD

Para ejecutar la imagen del sistema operativo VxWorks en la plataforma, se debe seguir los pasos encontrados en el archivo *target.txt*, que se encuentra en la carpeta de la instalación de VxWorks /WindRiver/VxWorks/21.07/os/unsupported/rpi_4/rpi_4/. Dichas instrucciones son específicamente para plataformas basadas en el procesador ARM Cortex-A72 de la Raspberry Pi 4. Para otras plataformas consulte de igual manera, los pasos contenidos en el archivo *target.txt* del embebido de preferencia. A continuación se muestran los pasos que se deben seguir:

1. Primeramente, se debe descargar el *firmware* para la Raspberry Pi 4, ya que este tiene el programa básico que permite al hardware del dispositivo funcionar sin problemas. Debe considerarse que el archivo del firmware pesa 30.9 GB aproximadamente. Para ello, se descarga en la máquina host utilizando el comando:

```
git clone https://github.com/raspberrypi/firmware.git
```

2. Como siguiente paso se debe moverse de *commit* del *git*, ya que se recomienda utilizar esa versión del firmware:

```
git checkout d9c382e0f3a546e9da153673dce5dd4ba1200994
```

3. Se formatea la tarjeta SD tipo FAT32.

4. Se copia solamente el contenido de la carpeta *boot* del firmware descargado (48.3 MB) e insertarlo en la tarjeta SD.

5. Se debe ejecutar U-boot en la Raspberry Pi, este permite arrancar la imagen ejecutable de ARM . Para ello se debe instalar U-boot que se encuentra en la carpeta llamada _sd_card_files, del directorio /WindRiver/VxWorks/21.07/os/unsupported/rpi_4/rpi_4/. Los archivos contenidos en dicha carpeta se deben copiar en la SD.

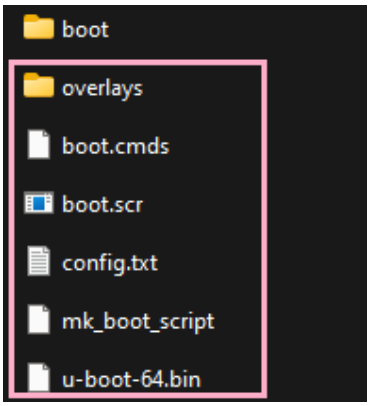

Figura 3.4: Archivos contenidos dentro de la SD.

6. Los archivos contenidos en *boot* deben colocarse fuera de dicha carpeta, en el mismo directorio que lo agregado en la Figura 3.4.

3.3.2. Crear una Imagen de VxWorks

3.3.2.1. Crear un VSB para la Raspberry Pi 4.

1. Generar un nuevo proyecto en: File → New → Wind River Workbencch Project.

2. En la ventana de *Build type*, seleccionar *VxWorks Source Build*. Seguidamente selecciona siguiente.

3. Se coloca el nombre para el VSB y se selecciona siguiente. Se mantiene la creación del proyecto en el *workspace*.

4. Se deben configurar los siguientes aspectos, para seleccionar la plataforma a utilizar. En la casilla "BSP" se debe buscar: *rpi_4_0_1_2_0 UNSUPPORTED*, y además en la casilla "Active CPU" seleccionar la *CPU CORTEX_A72*. En la sección de opciones, en la casilla de "Debug mode" seleccionar *On, and comiler optimizations disabled*, en la casilla "IP version setting" seleccionar *IPv6 and IPv4 enabled libraries* y en la casilla "VSB profile" seleccionar *DEVELOPMENT*. Finalmente, seleccione *Finish* para crear el VSB.

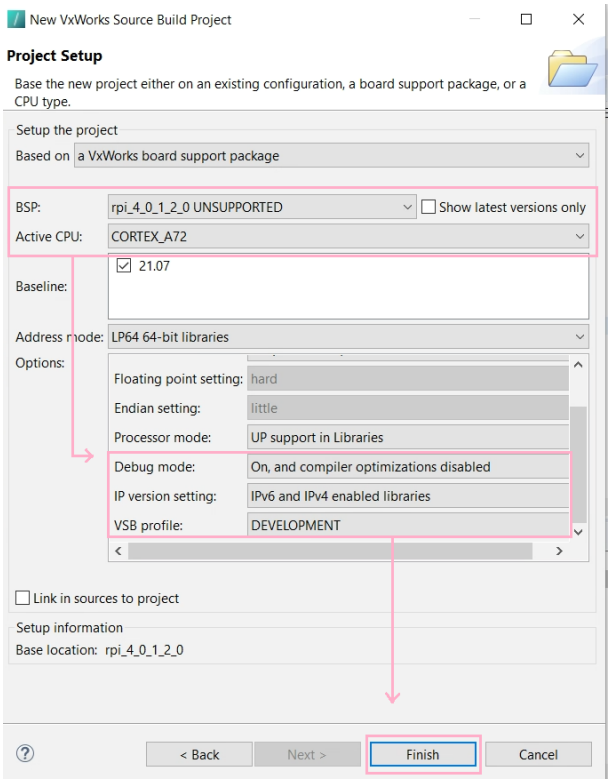

Figura 3.5: Configuración para generar un VSB para la Raspberry Pi 4 - Modelo B.

5. En la sección de *Project Explorer*, se puede observar el proyecto VSB creado, al desplegar los archivos que contiene, se debe dar doble clic en *Source Build Configuration*. Seguido, se observa la ventana de configuración del VSB. Al ejecutar *CTRL+F*, se debe asegurar que los siguientes componentes estén habilitados.

```
FS
FS_ROMFS
```

6. Se debe dar click derecho y seleccionar la opción *build* para construir el VSB.

3.3.2.2. Crear un VIP Basado en el VSB de la Raspberry Pi 4 modelo B.

1. Generar un nuevo proyecto en: File → New → Wind River Workbencch Project.
2. En la ventana de *Build type*, seleccionar *VxWorks Image Project*. Seguidamente selecciona siguiente.
3. Se coloca el nombre para el VIP, se selecciona opción para crear el proyecto en el *workspace* y se selecciona siguiente.
4. Se deben configurar los siguientes aspectos para poder utilizar el embebido Raspberry Pi 4. Debe estar basado en el VSB creado anteriormente, por lo que en la configuración de la casilla "project" se debe seleccionar el nombre del VSB creado. Además, en la casilla "BSP" se debe buscar *rpi_4_0_1_2_0*. En la sección de opciones se debe utilizar tal y como se muestra en la Figura 3.6. En la casilla de *Debug mode* seleccionar *On, and comiler optimizations disabled*. Finalmente, seleccione *next*.

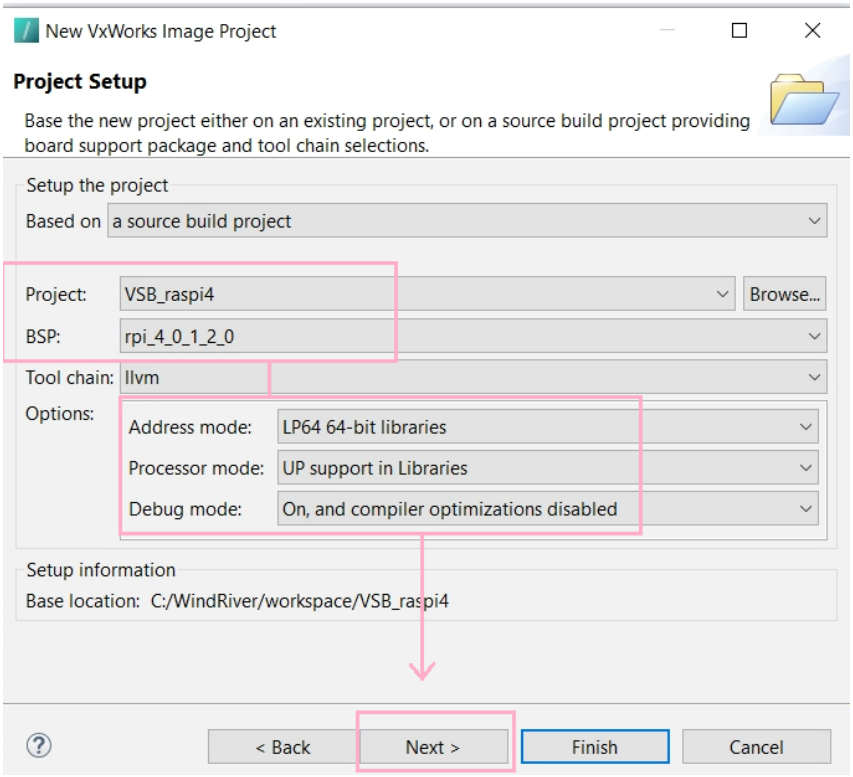

Figura 3.6: Configuración del VIP para la Raspberry Pi 4 - Modelo B.

5. En la siguiente ventana, se debe seleccionar en la casilla de "profile" el texto *PROFILE_RPI4*, y el botón de finalizar.

6. En la sección de *Project Explorer*, se puede observar el proyecto VIP creado, al desplegar los archivos que contiene, se debe dar doble click en Kernel Configuration. Seguido, se observa la ventana de configuración del VIP que cuenta con tres pestañas: **overview, bundles y components**.

7. En la pestaña *bundles*, para habilitar o deshabilitar componentes, se debe dar clic derecho y seleccionar *add*. Los paquetes recomendados son: BUNDLE_EDR (contiene los componentes requeridos por el kernel para detectar y reportar errores), BUNDLE_RTP_DEVELOP (componentes del kernel requeridos para desarrollar RTP), BUNDLE_RUNTIME_ANALYSIS_DEVELOP (componentes requeridos para el análisis en tiempo de ejecución), y BUNDLE_STANDALONE_SHELL (este paquete proporciona herramientas de *shell* del kernel con una tabla de símbolos independiente).

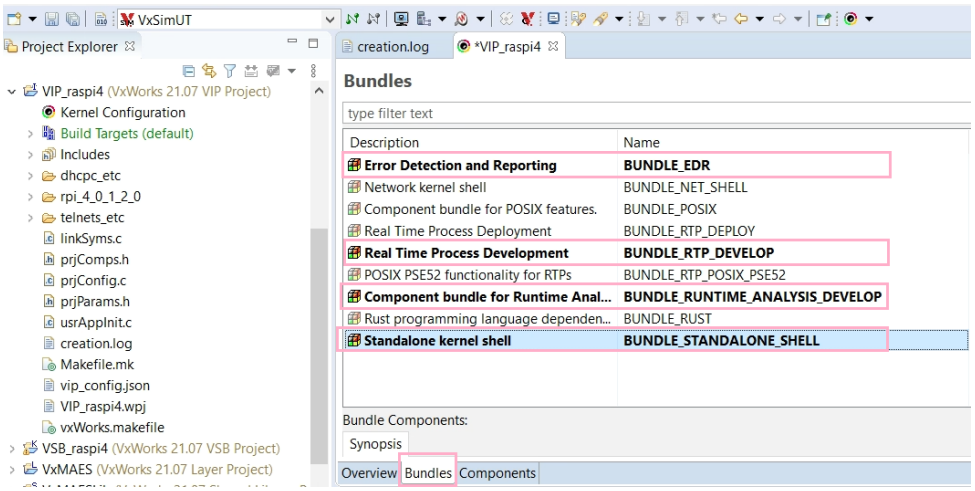

Figura 3.7: Componentes recomendados para el VIP

8. Se debe asegurar que los siguientes componentes estén incluidos para la tarjeta RPI4, los que no se encuentren incluidos, dar click derecho y seleccionar *include*. Para buscarlos componentes se utiliza el comando CTRL + F.

```
DRV_SIO_FDT_NS16550
DRV_SIO_FDT_ARM_AMBA_PL011
DRV_INTCTLR_FDT_ARM_GIC
DRV_ARM_GEN_TIMER_NG
DRV_FDT_BRCM_2711_PCIE
DRV_GPIO_FDT_BCM2711
DRV_I2C_FDT_BCM2711
DRV_PINMUX_FDT_BCM2711
DRV_SPI_FDT_BCM2711
INCLUDE_USB_XHCI_HCD_INIT
DRV_END_FDT_BCM_GENETv5
DRV_FDT_BRCM_2711_EMMC2
INCLUDE_USB_GEN2_STORAGE_INIT
INCLUDE_DOSFS
INCLUDE_XBD_PART_LIB
DRV_SDSTORAGE_CARD
```

9. Otros componentes VIP que son recomendados para el VIP tanto para la RPI4 y si se utiliza el simulador VxSim. Al finalizar, utilice el comando CTRL + S, para guardar los cambios.

```
INCLUDE_USB
INCLUDE_USB_INIT
FOLDER_USB_HOST_DEVICES
INCLUDE_EVDEV_LIB_KBD_SHELL_ATTACH
INCLUDE_PING
INCLUDE_IFCONFIG
FOLDER_IPCOM_SHELL_CMD
INCLUDE_ROMFS_DRV
```

10. En la pestaña de *Project Explorer*, dar click derecho en la carpeta del VIP, y seleccionar la opción *build* para construir el VIP.

11. En la ventana de *Project Explorer*, en la carpeta *default* del proyecto VIP creado, se encuentra el archivo **rpi-4.dtb** y **uvWorks**, es el árbol de dispositivos, que permite que el SO pueda utilizar los componentes de hardware, y la imagen de VxWorks, respectivamente. Estos deben ser agregados en la tarjeta SD.

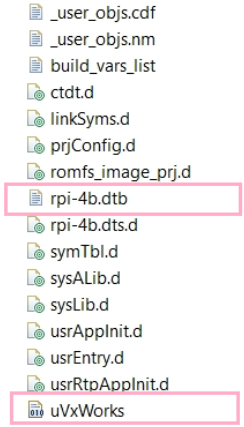

Figura 3.8: Imagen del SO VxWorks generado por el VIP.

3.3.3. Archivo ROMFS para Imagen VIP

Antes de iniciar los pasos de este punto deben generarse las secciones de generar una biblioteca y las aplicaciones RTP de la sección 3.2, pero se debe asegurar que al generar las aplicaciones deben basarse en el VSB de la Raspberry Pi 4.

Además, también necesario seguir la sección del capítulo 5 llamado "Configuring and Building a ROMFS File System Project" de la guía "VxWorks Kernel Application and RTP Application Projects Guide - 21.07". Esta guía se encuentra en la dirección de instalación del Workbench: */WindRiver/vxworks/21.07/docs/Core/*.

Los archivos ROMFS permiten almacenar las bibliotecas necesarias para la ejecución de las aplicaciones, estos proyectos deben agregarse al VIP que se ha generado para la Raspberry Pi 4. Los archivos que debe contener el ROMFS se encuentran en:

- **VxMAESBib.so**: se encuentra en el directorio:

```
installDir/workbench-4/workspace/NombreVSB/VxMAESBib/Debug/VxMAESBib.so
```

- **libc.so.1**: se encuentra en el directorio:

```
installDir/workbench-4/workspace/NombreVSB/usr/root/NombreCompilador/bin/libllvm.
```

- **libllvm.so.1**: se encuentra en el directorio:

```
installDir/workbench-4/workspace/NombreVSB/usr/root/NombreCompilador/bin
```

- **Archivo.exe** se encuentra en el directorio:

```
installDir/workbench-4/workspace/NombreRTP/NombreCompilador/NombreRTP/debug/
```

Finalmente, el contenido de ROMFS se muestra en la Figura 3.9. Seguidamente, se debe de construir de nuevo la imagen VIP para que estos cambios sean reflejados en la imagen del SO para integrarlo a la microSD.

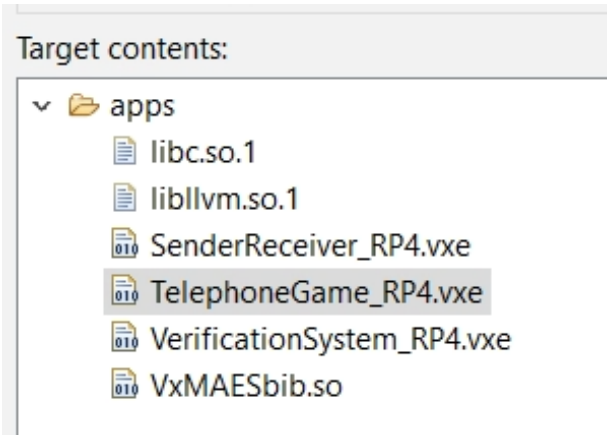

Figura 3.9: Contenido dentro del sistema de archivos ROMFS.

Seguidamente, se debe de construir de nuevo la imagen VIP para que estos cambios sean reflejados en la imagen del SO para integrarlo a la microSD. Los archivos dentro del SD, deben coincidir con la Figura 3.10

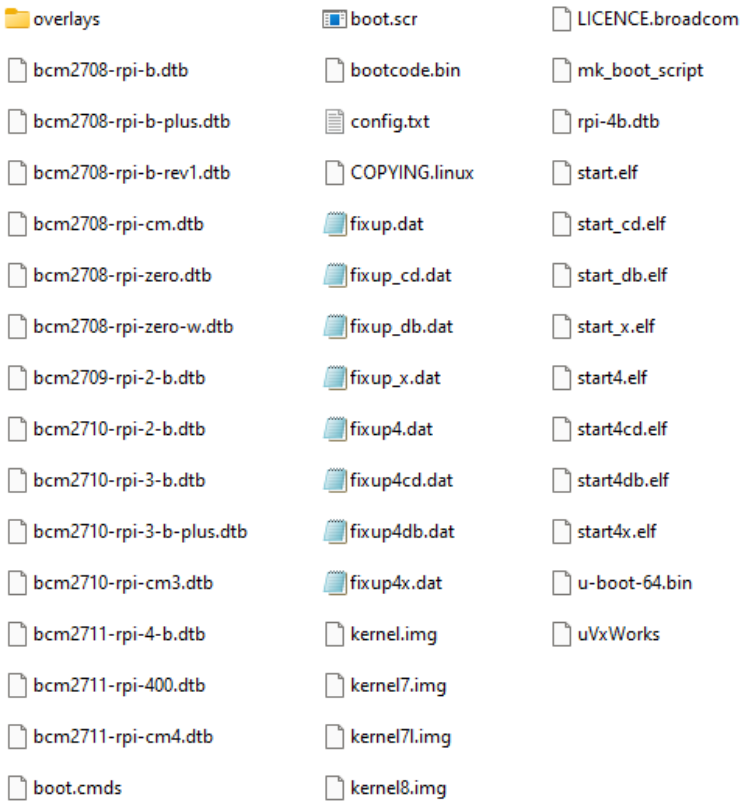

Figura 3.10: Archivos contenidos en la microSD.

3.3.4. Implementación de VxMAES en la Raspberry Pi 4 - Modelo B

La Raspberry Pi 4 utilizada cuenta con una memoria RAM de 2 GB y un procesador ARM Cortex-A72 de cuatro núcleos a frecuencia de 1.5 GHz. Tiene soporte para insertar micro SD en el que se coloca el sistema operativo o archivos. Al tener incorporado el SO en la microSD, se procede a visualizar el arranque del sistema. Ya que en VxWorks la Raspberry Pi 4 no tiene soporte para puerto HDMI, se debe acceder a la terminal del dispositivo mediante un convertidor serial FT232. El transmisor se debe conectar al pin 8 (GPIO 14), el receptor al pin 10 (GPIO 15) y la tierra en el pin 6. Se utiliza la plataforma PuTTY como medio para observar los resultados, verificando que el puerto serial COM corresponda al de la Raspberry Pi 4 y la tasa de baudios sea de 115200. En la Figura 3.11 se muestra la conexión entre el embebido Raspberry Pi 4 y el FT232.

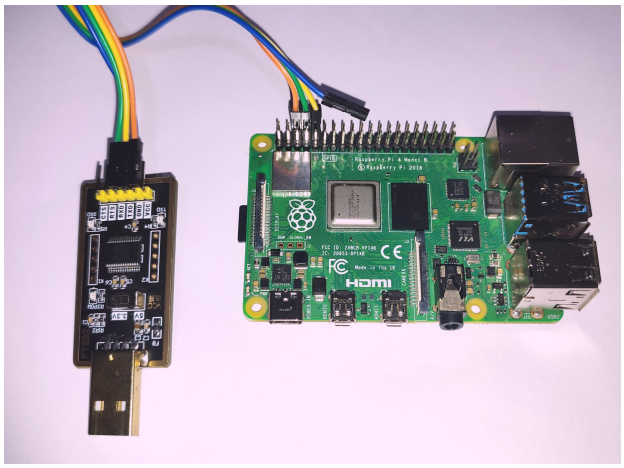

Figura 3.11: Conexión serial entre la Raspberry Pi 4 y el FT232.

Cuando se conecta el dispositivo embebido a una fuente de alimentación de 5V @ 3A, se mostrará en pantalla la información que se presenta en la Figura 3.12.

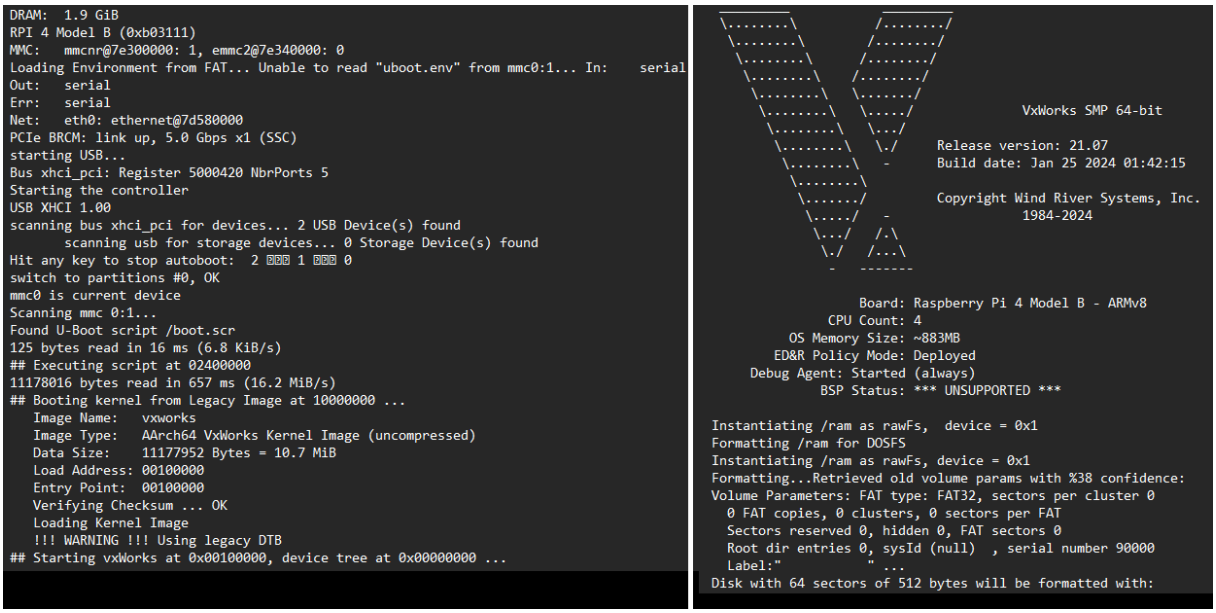

Figura 3.12: Pantalla de inicio VxWorks en Raspberry Pi 4.

Para ejecutar las aplicaciones se debe cambiar al interpretador de comandos de VxWorks con el comando **cmd**, y dirigirse a la carpeta donde están integradas las aplicaciones (carpeta romfs).

```
[vxWorks *]# cd /romfs
[vxWorks *]# ls
```

Estas se ejecutan utilizando el comando:

```
rtp exec NombreRTP.vxe
```

Bibliografía

[1] C. Chan Zheng, "MAES: A Multi-Agent Systems Framework for Embedded Systems", M.S. Thesis, Delft University of Technology, Países Bajos, 2017. [En línea]. Disponible en: https://repository.tudelft.nl/islandora/object/uuid%3Ae1c076dc-ab64-4e77-8453-bce618033907

[2] D. Rojas, "FreeMAES: Desarrollo de una biblioteca bajo el Paradigma Multiagente para Sistemas Embebidos (MAES) compatible con la NanoMind A3200 y el kernel FreeRTOS", B.S. Thesis, Ing. en Electrónica, Instituto Tecnológico de Costa Rica, Cartago, 2021.

[3] A. Conejo, "CMAES: Implementación de la biblioteca FreeMAES basada en el Paradigma Mutliagente para Sistemas Embebidos en la computadora espacial NanoMind A3200 utilizando el sistema operativo en tiempo real FreeRTOS ", B.S. Thesis, Ing. en Electrónica, Instituto Tecnológico de Costa Rica, Cartago, 2022.