Communication Architecture Design and Case Study of Embedded Partition Real-Time Operating System

Penghui Ren
School of Computer Science and Engineering
Xi'an Technological University
Xi'an, 710021, China
E-mail: rph_0290@163.com

***Abstract*—With the continuous development of integrated modular avionics system, a large number of applications have higher and higher requirements for the operating system. However, the kernel and application process of traditional embedded real-time operating system often run at the same privilege level. A wrong operation may cause the normal operation of the whole kernel or other processes, resulting in system crash, Embedded real-time partition operating system is widely used because of its good protection ability of time partition and space partition. Because partitions are isolated, in order to carry out data communication between partitions, it is necessary to adopt the way of inter partition communication for information transmission. This paper introduces the architecture of partitioned embedded operating system, discusses the communication principle and design process between partitioned modules, and focuses on the communication mechanism of sampling port and queue port. In addition, a communication mechanism based on virtual port is used to solve the problem that the port bound by the application process in a partition cannot communicate with the communication equipment between the partition module and other partition application processes. Finally, the design process of socket communication in partitioned operating system and the sending and receiving process of data under partitioned operating system are proposed.**

Keywords-Partition Operating System; Communication Between Modules; Virtual Port; Socket Communication

I. INTRODUCTION

With the rapid development of science and technology in China and the miniaturization and specialization of embedded operating system, embedded operating system is developing from a relatively single weak function to a more professional strong function. Embedded real-time operation system (RTOS) is the core software of airborne equipment. It is widely used by foreign enterprises and companies because of its small kernel, high stability, strong real-time and tailorability. However, in these generally applicable embedded operating systems, such as VxWorks, wince, deltaos, etc., the application process and the kernel are in the same operating system at the same time. Therefore, the wrong call of an application process may cause the wrong response of the kernel or other application processes, resulting in the failure of the system to run normally. Therefore, in order to protect the system resources and avoid the impact between applications with different functions or security levels, it is necessary to independently develop an embedded real-time operating system with its own independent address space and no mutual influence in time cycle.

ARINC653 is the main operating system specification to meet the requirements of integrated avionics real-time operating system. The most important thing is to put forward the concept of partition [1]. Partition refers to the collection of two or more application processes with similar or related functions running on the same processor module. The implementation of partition mainly includes time partition and space partition. Spatial partition means that each partition in the operating system has its own independent address space. By using the storage manager to establish the mapping between the partition address and the actual physical address for each partition, each partition has its own independent and unique storage address to ensure that all partitions in the space are independent of each other. Time partition means that each independent partition is scheduled in rotation according to a specific cycle. The priority of each partition is the same. The operating system provides a fixed time length, which can be divided into multiple time fragments. In this fixed time, Each partition will be allocated at least one time fragment, and the partition can only be accessed within the allocated time fragment, so as to ensure the independence and correctness of application processes in each independent partition[7].

The application process of partitioned operating system is isolated from each other in time and space, so the communication between partitioned modules has become the main way of data exchange between partitions. Section 1 introduces the architecture of partitioned embedded operating system; Section 2 describes the related contents of inter partition communication, including the concept of inter partition communication, the communication mechanism of sampling port and queue port; In Section 3, a simple design of port communication between partition modules is carried out; Section 4 mainly discusses and designs the working principle, interface function and data sending and receiving process of TCP socket under partitioned operating system; Section 5 summarizes.

II. PARTITION EMBEDDED REAL-TIME OPERATING SYSTEM

A. System architecture

The software structure of partitioned operating system is usually divided into three layers, including application layer, operating system layer and hardware module support layer.

The application layer is a partition application developed by users and runs on the operating system.

The operating system layer mainly implements hardware independent functional services, including the basic core functions of the operating system and various configurable components to meet the needs of specific applications. The operating system also includes partition operating system and core operating system. The partition operating system is the manager of resources in the partition to realize process management, scheduling and resource allocation in the partition. The core operating system mainly realizes partition management, scheduling, inter partition communication, system fault monitoring, resource management and equipment management in the system.

The basic core of partition embedded real-time operating system provides general control services of real-time operating system, including task management, inter partition communication, interrupt / exception management, clock / timer management, cache management, user expansion, error handling, health monitoring, storage management, device management, virtual file system management and other functions.

Configurable components are components that provide specific functional requirements for different airborne software, mainly including C runtime library, VxWorks compatible interface, bit management and file system module.

The module support layer is composed of specific hardware module support software developed according to specific interface specifications, which mainly realizes the isolation between hardware and operating system layer. The module support layer mainly includes structure support package, board level support package, MSL layer debugging agent and image management.

The API for the interaction between the operating system layer and the module support layer interface is agreed by both of them. The module support layer supports the hardware support services required by the operating system layer.

The architecture of partitioned embedded real-time operating system is shown in Figure 1[2].

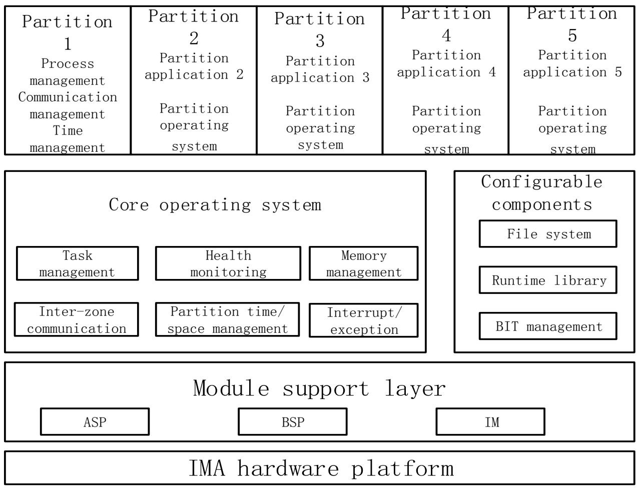

Figure 1. Partition embedded real-time operating system architecture

B. Process management

Process management is mainly responsible for the creation, scheduling and deletion of all processes in the partition. There can be two types of processes in the partition at the same time, namely, periodic processes executed at a fixed frequency and aperiodic processes triggered by events. The process states include ready state, running state and waiting state. The basic state and its changes are shown in Figure 2.

Any process can be preempted by other processes in this partition at any time. When the partition is activated, the process in the ready state is executed. Processes can lock some programs through preemption control mechanism, that is, CPU resources will not be preempted by other processes in the partition until they are unlocked. If a protected locked process in the partition is interrupted due to the end of the partition time window, ensure that the process is executed when the partition time window arrives again.

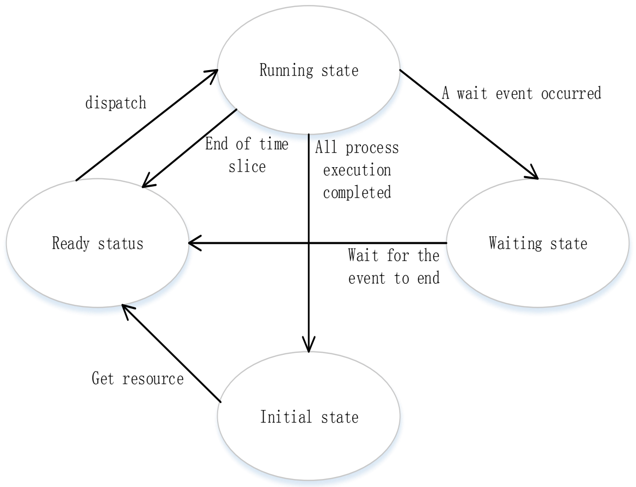

Figure 2. Process basic state and its transition diagram

C. Design objectives

The partition architecture design of separated kernel is to design the partition operating system from three aspects: partition isolation, reducing coupling and adding an intermediate layer.

1) Partition isolation

Considering the reliability design of separated kernel, the basic unit of embedded real-time operating system is task, and the resources occupied by a task are memory space and CPU time. Therefore, the kernel can be isolated from these two aspects, and different tasks can be placed in the partitions that have been isolated in time and space, so that they do not affect each other. Because each task runs in its own different partition, it does not interfere with other tasks, so as to enhance the reliability of the system.

2) Add intermediate layer

David Wheeler, a famous British computer scientist, once said the famous saying "All problems in computer science can be solved by another level of indirection." it means that all problems in the computer field can be solved by adding an indirect middle layer. The idea of adding an intermediate layer is to consider the architecture of embedded real-time operating system and provide the reliability of RTOS by adding an intermediate layer.

3) Reduce coupling

Low coupling is an important design pattern idea. On the one hand, low coupling reduces the range of other modules affected by the change of one module; On the other hand, it makes the module more cohesive, simpler structure, stronger tailor ability and easy to understand. The idea of low coupling in the design of zoning system will help to improve the reliability of the system. Taking VxWorks as an example, its wind kernel contains functions such as task management, synchronous communication and memory management. These functions are often cohesive with the device driver. The error of any module will lead to system crash. The reason is that the coupling between RTOS kernel function modules is too high, which leads to the reduction of its reliability. Therefore, the idea of low coupling is adopted to split each functional module and reorganize the RTOS structure to make it independent, so as to enhance its reliability.

In short, the core idea of partition system design is "isolation". This idea is reflected in the partition strategy of space-time isolation, the middle layer of implementation isolation, or the function block isolation to reduce coupling.

III. INTER MODULE COMMUNICATION

Communication between partition modules [8], that is, data exchange between partition modules. The only way of communication is through messages, ports and channels. The communication between partition modules is completed by sending and receiving messages through ports. Messages are sent from one source port to one or more destination ports. The port is visible to the user. Channel provides the interconnection mechanism between ports. Each channel indicates the port name and partition of sending messages, as well as the port and partition of receiving messages. The relationship between port and channel is mapped through XML configuration file. When using ports in the module, first call the port creation service to complete the creation of port objects in the partition module and realize the connection with core communication resources. The partitions communicating with each other can be in the same processor module or in different processor modules. The channel defines the logical relationship between a source port and one or more destination ports, and also defines the transmission mode and characteristics of messages from the source port to the destination port.

The communication service function between partition modules provided by the partition operating system that complies with the ARINC653 standard, on the basis of meeting the communication function between partition modules on the same module, also needs to provide support for the ability to communicate between partitions on different modules [3].

There are two types of communication services between partition modules: one is sampling mode, and the other is queue mode. The sampling mode is suitable for transmitting data messages that are generally similar and constantly updated. There is only one valid message buffer in the system, and the message remains in the buffer until it is overwritten by a newly sent message. Each module of the zone can send messages to the sampling source port at any time, or access the destination port information at any time. The queue mode is suitable for transmission that contains different data information, and does not allow the message to be overwritten, and the message is generally not allowed to be lost. The message remains in the source port until it is sent successfully, or it remains in the destination port until it is successfully received by the application port.

A. Sampling port message communication

The message of the sampling port [5] does not provide a queuing mechanism, that is to say, the sending and receiving operations will not suspend the user process, there is one and only one effective message buffer in the system, and the newly sent message will overwrite the previously sent message. The communication process of the sampling port message is shown in Figure 3.

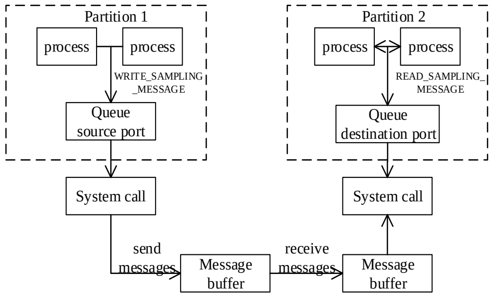

Figure 3. Sampling port message communication process

For the sender, the process calls the WRITE_SAMPLING_MESSAGE service to initiate a request to write a sampled message. At this time, the port service is checked for legitimacy, including the port ID and the legitimacy of the message. After it is legal, the user sends data to the port. If the port has no data at this time, the data is copied to the port's buffer for data transmission; if there is data in the port at this time, the original data is overwritten.

For the receiving end, the process needs to call the READ_SAMPLING_MESSAGE service to initiate a request to read the sampled message. At this time, the legality of the service (port ID and other parameters) of the destination port needs to be checked. After it is legal, the user starts to receive data from the receiving port. If there is no message on the port at this time, the message is copied to the receiving port buffer, and then the current port is empty; if there is a new message at this time, the original old message Cover, and then calculate the age of the message based on the current time and the time when the message arrives at the port, determine whether the message is valid, and finally return the validity of the message to the user.

B. Queue port message communication

The message of the queue port[5] supports a queue waiting mechanism. The operating system will create a limited message queue depth according to the situation of the message queue, maintain the state of the source port and the destination port, and then determine the processing operations that need to be performed according to the state of each port. The communication process of the queue port message is shown in Figure 4.

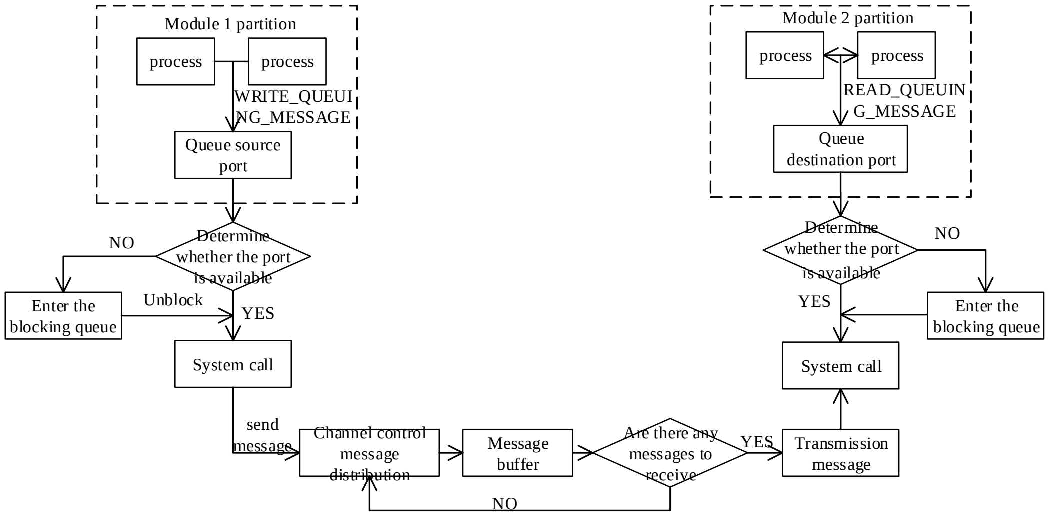

Figure 4. Queue port message communication process

For the sender, the user process calls the WRITE_QUEUING_MESSAGE service to initiate a request to send a queue message. At this time, it starts to check the legitimacy of the port, and then determines the processing strategy according to the state of the source port. If the sending port does not have a free buffer, the port is in an unavailable state at this time, and the process will enter the blocking queue; if the sending port has a free buffer, that is, the current port is in an available state, the system calls the channel control program to carry out the message distribution. The message to be sent is copied to the message buffer, and the channel control program completes the message sending operation according to the mutual connection between the source port and the destination port.

For the receiving end, the user process calls the READ_QUEUING_MESSAGE service to receive messages from the receiving port, and at the same time checks the legitimacy of the port, and then determines the subsequent operation according to the state of the destination port. If there is no message at the destination port at this time, the process will enter the blocking queue of the receiving message port; if there is a message, it proves that the current port is available, and the message is copied from the message buffer to the message queue of the corresponding destination port through a system call. Until all the destination ports have received the message, the system buffer is notified to release the space of the corresponding message buffer and the message queue of the source port.

IV. PORT COMMUNICATION DESIGN BETWEEN PARTITION MODULES

A. Port communication between partitions

In the partitioned operating system, the ports have the following types: the first is a local port, which allows the application process to communicate with other application processes on the same module, and the local port is attached to the partition of the module; the second is Virtual port, through the virtual port can communicate with the partition outside the partition module, by connecting a port to the underlying driver; the third type is direct access to the port, this port implements a queue port without a software buffer, and it can also It is directly used to communicate with the partition outside the module. However, a channel with a direct access port must have a source address and a destination address. The direct access port can be in a partition or other partitions.

The communication between partition modules, as the name suggests, is data exchange between different partition modules, which determines that the sending partition and the receiving partition will not be in the same operating system, so the concept of virtual port [4] is introduced.

Corresponding to the real port we are talking about, we can regard the virtual port as a temporary port in this module of the external module that communicates with the real port, but ultimately the message transmission between modules is completed through the underlying driver. The communication link between the partition modules is shown in Figure 5.

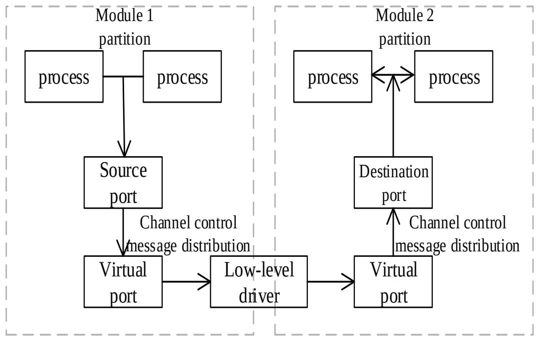

Figure 5. Communication between partition modules

For the sender, the destination port is outside the module, so we configure a virtual port for the destination port, which corresponds to the external device. After the user process sends data to the destination port, the operating system calls the underlying driver through the virtual port to send the data.

For the receiving end, the source port is outside the module, so we configure a virtual port for the source port. The virtual port calls the underlying driver to receive data, and then passes the received data to the user process through the destination port.

B. Virtual port

1) Virtual port function

The virtual port is a logical structure that connects the partition application port and the lower-layer network drive device. It connects the upper-layer source port and destination port through a channel. The function prototype is as follows.

```
STATUS portVirtualDrvAdd
(PORT_DRV_FCT *pPortDrvFct,
unsigned char *name)
```

Name is the name of the connected underlying drive device. The PORT_DRV_FCT structure contains 6 function pointers for virtual port creation (createRtn), virtual port read data (readRtn), virtual port write data (writeRtn), virtual port status acquisition (statusRtn), virtual port for port attachment (attachRtn) and virtual port validity check (availableRtn).

During the initialization of the inter-area communication resources, for the virtual port, the virtual port object is first attached to the virtual port driver by calling attachRtn. The input parameter of the attachRtn function is the related information of the virtual port object, and the output parameter is the ID assigned to the virtual port object by the attached virtual port driver. Pass this ID into the operating system as an input parameter when calling createRtn, readRtn, writeRtn, statusRtn, and availableRtn functions.

2) The logical structure of the virtual port

For the sending end, the virtual port logical structure table is used to record the information of all sending ports in the zone and provide information about the sending port of the zone.

PortID is the ID of the sending port; MsgName is the topic information of the current port; MsgMaxSize represents the maximum buffer message size of the sending port Length; MsgMaxNum represents the maximum number of buffered messages; MsgQueueID is the message queue ID of the buffered message; DestMsgQueueID represents the ID of the buffer message queue of the receiving port bound to the sending port. There can be multiple message queue IDs. When DestMsgQueueID is 0 When represents that the sending port is not bound to any receiving port; PsudoID represents the user configuration number of the current virtual port; DevHdr represents the device handle of the underlying driver port; EmptyFlag is a flag for judging whether the current virtual port message buffer is empty. The core operating system sends the data of the message queue of the sending port buffer to the message queue of the receiving port according to the information of the sending port and the virtual port.

For the receiving end, the logical structure table of the virtual port is used to record all the receiving port information in the partition and provide the information of the receiving port in the partition. PortID is the ID of the receiving port; MsgName is the subject information of the receiving port; MsgMaxSize represents the maximum length of messages buffered by the receiving port; MsgMaxNum represents the maximum number of buffered messages; MsgQueueID is the message queue ID of the receiving port buffer; IsAssign represents whether the port is Bind to the sending port, 1 means binding, 0 means unbound. The core operating system binds the virtual port and the port according to the information of the sending port and the information of the receiving port to establish a data channel for communication. The partition application receives data from the port buffer based on the PortID.

The logical structure of the virtual port is shown in Table 1.

TABLE I. LOGICAL STRUCTURE OF VIRTUAL PORT

content	Function
PortID	Port ID (receive/send, the same below)
MsgName	Subject information
MsgMaxSize	Maximum length of buffered message
MsgMaxNum	Maximum number of buffered messages
MsgQueueID	Message queue ID
DestMsgQueueID	The ID of the buffer message queue of the receiving port bound to the sending port (sending)
PsudoID	Virtual port user configuration number
DevHdr	Device handle of the underlying driver port
EmptyFlag	Whether the message buffer is empty
IsAssig	Whether to bind with the sending port (receive)

V. COMMUNICATION EXAMPLE

A. Socket overview[6]

Socket provides three types of sockets, namely streaming sockets, datagram sockets and raw sockets. Streaming sockets provide a connection-oriented, reliable data transmission service. The data is sent without errors and repetitions, and is received in the sending order, using the TCP protocol. The datagram socket provides a connectionless service. The data packet is sent in the form of an independent packet without error-free guarantee. The data may be lost or duplicated, and the receiving sequence is disordered. The UDP protocol is used. Raw sockets are often used to test the implementation of new protocols or access new devices configured in existing services. This interface allows direct access to lower-level protocols such as IP and ICMP. This article mainly introduces the streaming socket mode.

B. Socket layer design

In each partition operating system, the Socket layer completes the communication between data by calling the Socket interface [9] provided by the kernel TCP/IP protocol stack [11]. The specific interfaces and functions used are shown in Table 2.

TABLE II. INTERFACE FUNCTION AND CORRESPONDING FUNCTION

Interface function	**Function**
socket( )	Create a socket descriptor
bind( )	Bind the socket to a specific TCP port
listen( )	Listening socket
connect( )	Send a connection request to the server (client-only)
accept( )	Accept connection request (server exclusive)
send( )	send data
recv( )	Receive data
close( )	Close socket

1) Create socket

Function prototype:int socket(int domain,

int type,int protocol);

Create a socket to complete the following tasks:

a) Set protocol family;

b) Specify the socket type;

c) Specify the protocol related to the socket type.

2) Bind socket

Function prototype: int bind (int sockfd, const struct sockaddr *addr, socklen_t addrlen);

The main task of binding a socket is to assign a specific address (ip address + port number) in an address family to the socket.

3) Listening socket

Function prototype: int listen (int sockfd, int backlog);

The created socket is of an active type by default. The task to be completed by the monitoring socket is to change the socket to a passive type and wait for the client's connection request.

4) Send connection request

Function prototype: int connect (int sockfd, const struct sockaddr *addr, socklen_t addrlen);

The client establishes a connection with the TCP server by calling the connect function.

5) Accept connection request

Function prototype: int accept (int sosckfd, struct sockaddr *addr, socklen_t *addrlen);

After the TCP server listens to the connection request sent by the client, it will call the accept function to receive the request, thereby successfully establishing the connection. After that, the network I/O operation is started.

6) Data sending

Function prototype: ssize_t send (int sockfd, const void *buf, size_t len, int flags);

Data transmission completes the following tasks:

a) Receive the data submitted by the virtual port into the send buffer;

b) Determine the destination IP address;

c) Determine the destination port;

d) Send data。

7) Data reception

Function prototype: ssize_t recv (int sockfd, void *buf, size_t len, int flags);

Data reception completes the following tasks:

a) Receiving data on the bound TCP port;

b) Submit the received data to the virtual port first, and then submit it to the application buffer through the virtual port buffer.

8) Close socket

Function prototype: int close (int socketfd);

After the client and the server complete the data receiving and sending operations, the corresponding socket descriptor is closed, and the descriptor can no longer be used by the calling process.

After the partition operating system is started, task A of partition 1 enters the processing and waiting, and waits for the response of the network card driver and the IPC messages from other partitions in turn. The tasks of other partitions start to establish sockets to prepare for network communication. First call v_socket, v_bind and other interfaces to establish a socket connection, and then call v_listen or v_connect to monitor or establish a connection. If the application is a server, after listening to the request information Call v_accept to accept the request, send a socket command request to the task of partition 1 through SYN_SEND, call the v_recv/v_send function for network communication, and call SYN_RECV to wait for the result returned by task A. Other tasks of partition 1 receive service requests such as v_socket and v_bind from other partitions and are activated to create socket devices and establish communication, and then send and receive network data with other partitions, and then send socket handles and network addresses and other information Reply to a certain task B of other partitions through SYN_SEND, and the socket communication enters the ready state at this time. Task B sends and receives network information through the socket. All the processes are the same as the execution of task A. After receiving the reply from task B, a socket communication is completed.

C. Data sending and receiving process[10]

1) Data sending process

When sending data to the receiving end, the application process first obtains a socket, searches for the corresponding virtual port according to the IP address and port number bound to the socket, and then determines whether the connection is a TCP connection. If so, submit the data to be sent to the virtual port, and then call the send() function to send the data on the virtual port to the virtual port of the receiving end, Finally, judge whether the transmission is successful. If successful, return a parameter value of successful transmission. The process is shown in Figure 6.

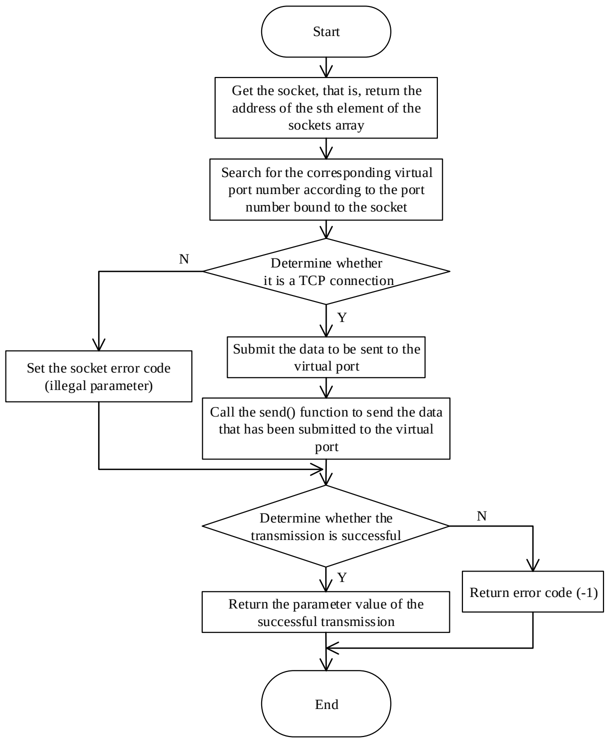
Start
Get the socket, that is, return the address of the sth element of the sockets array
Search for the corresponding virtual port number according to the port number bound to the socket
Determine whether it is a TCP connection
N
Y
Set the socket error code (illegal parameter)
Submit the data to be sent to the virtual port
Call the send() function to send the data that has been submitted to the virtual port
Determine whether the transmission is successful
N
Y
Return the parameter value of the successful transmission
Return error code (-1)
End

Figure 6. Data sending process in partitioned operating system

2) *Data receiving process*

When the application process receives data from the sender, it first obtains the socket, that is, selects the address of an element from the sockets array, returns its socket structure, and judges whether the connection is a TCP connection. If so, according to the socket binding Search for the corresponding virtual port number for the IP address and port number, then call the recv() function to receive data from the sender, and then determine whether the data is received, if it is received, submit the received data to the virtual port buffer, and finally the virtual port The data on the port is transferred to the port bound to the socket to complete the data reception. The process is shown in Figure 7.

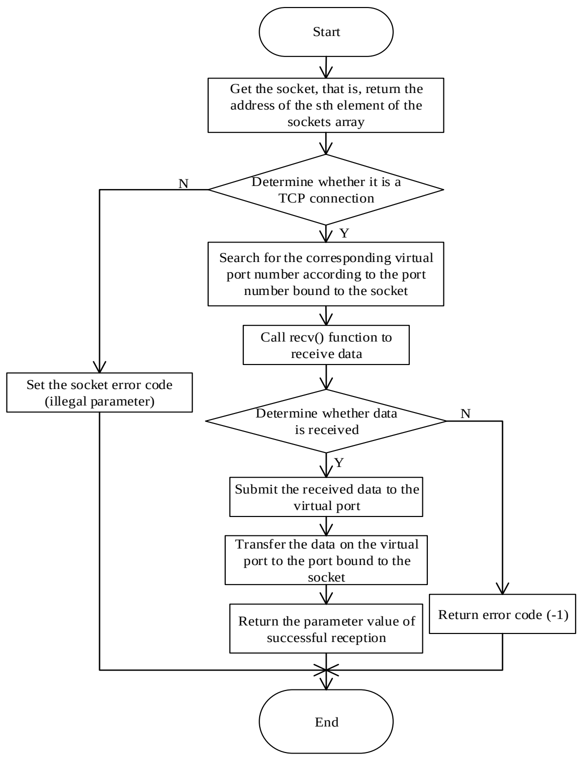
Start
Get the socket, that is, return the address of the sth element of the sockets array
Determine whether it is a TCP connection
N
Y
Search for the corresponding virtual port number according to the port number bound to the socket
Call recv() function to receive data
Set the socket error code (illegal parameter)
Determine whether data is received
N
Y
Submit the received data to the virtual port
Transfer the data on the virtual port to the port bound to the socket
Return the parameter value of successful reception
Return error code (-1)
End

Figure 7. Data receiving process in partitioned operating system

VI. SUMMARIZE

Through the analysis of the embedded real-time partition operating system architecture, this paper discusses the process management mode in the partition system and the design goals of the partition operating system, introduces the communication principle between partition modules, and focuses on the message of the sampling port and the queue port. Communication principle, and then design the port communication process between the partition modules, and realize the port communication by introducing the virtual port. At the same time, the virtual port function and logical structure are introduced in depth, and then the actual TCP socket layer of the partition operating system is introduced. The communication process is designed, using the basic function interface used by the traditional operating system, and finally a process of sending and receiving data under the partition operating system is proposed. Through an example to test whether the socket communication based on the virtual port mechanism of the partition operating system can be carried out, a simple communication result is obtained. In the future research work, we will continue to optimize the communication process of this design and the process of sending and receiving data. Although the communication between partitions has many advantages, if we do not understand and avoid the risks and problems that may be caused by the communication between partitions, it will inevitably bring many unforeseen problems in the future design work. Therefore, it is necessary to standardize and strictly design the communication process between the partition modules to reduce the risk, so that it can provide users with more powerful communication support.

REFERENCES

[1] Tao Yongchao, Song Qilong, Piao Songhao. Design and implementation of partition Operating System based on ARINC653 standard [J]. Journal of Physics: Conference Series, 2021, 1732(1).

[2] Tong Yan, Yuan Haofang, Xu Fei, Wu Zhiming, Wang Manda. Research on partition operating system based on ARINC653 [J]. Electronic Testing, 2020(13).

[3] Xu Xiaoguang, Ye Hong. Design and Implementation of Interval Communication in Avionics System [J]. Aeronautical Computing Technology, 2005.

[4] Zhang Ming, Zhou Lin.Design and implementation of IMA based on VxWorks653 partition operating system [J]. Firepower and Command Control, 2014.

[5] Xu Xiaoguang, Yun Haishun, Xing Liang. The design of inter-partition communication under partition operating system [J]. Modern Electronic Technology, 2013.

[6] Zhang Xiaona, Chang Leran, Wu Wei, Liao Jinwei, Shen Liwen. Realization of Socket Communication under Linux System [J]. Electroacoustic Technology, 2020.

[7] Huang Runlong, Shen Qian, Gou Xiantai. Task scheduling of ARINC653 multi-core and multi partition operating system [J]. Telecommunications technology, 2020, 60(09):1108-1113.

[8] Yang juping. Research on partition technology based on embedded real-time operating system [J]. Industrial control computer, 2015, 28(05):29-30.

[9] Xiao Lei. Network communication design based on socket under VxWorks [J]. Computer and network, 2013, 39 (12): 66-68

[10] Xing Liang, Zhao Yi. Application design of socket communication under partitioned operating system [J]. Aviation computing technology, 2011, 41(05):88-90.

[11] Wang Xiaopeng. Socket and Winsock communication mechanism under TCP / IP [J]. Aviation computing technology, 2004 (02):126-128.