LR5-LAN Socket Communication

Sample Program

(Windows C++)

content

1. Overview

This is an outline of sample programming to control LR5-LAN via socket communication.

The programs are intended to control the unit using Microsoft Visual C++ control without using the DLLs provided by PATLITE.

1.1. System Overview

The system configuration diagram of this program is as follows.

The sample program controls one LR5-LAN by socket communication.

2. Development Environment

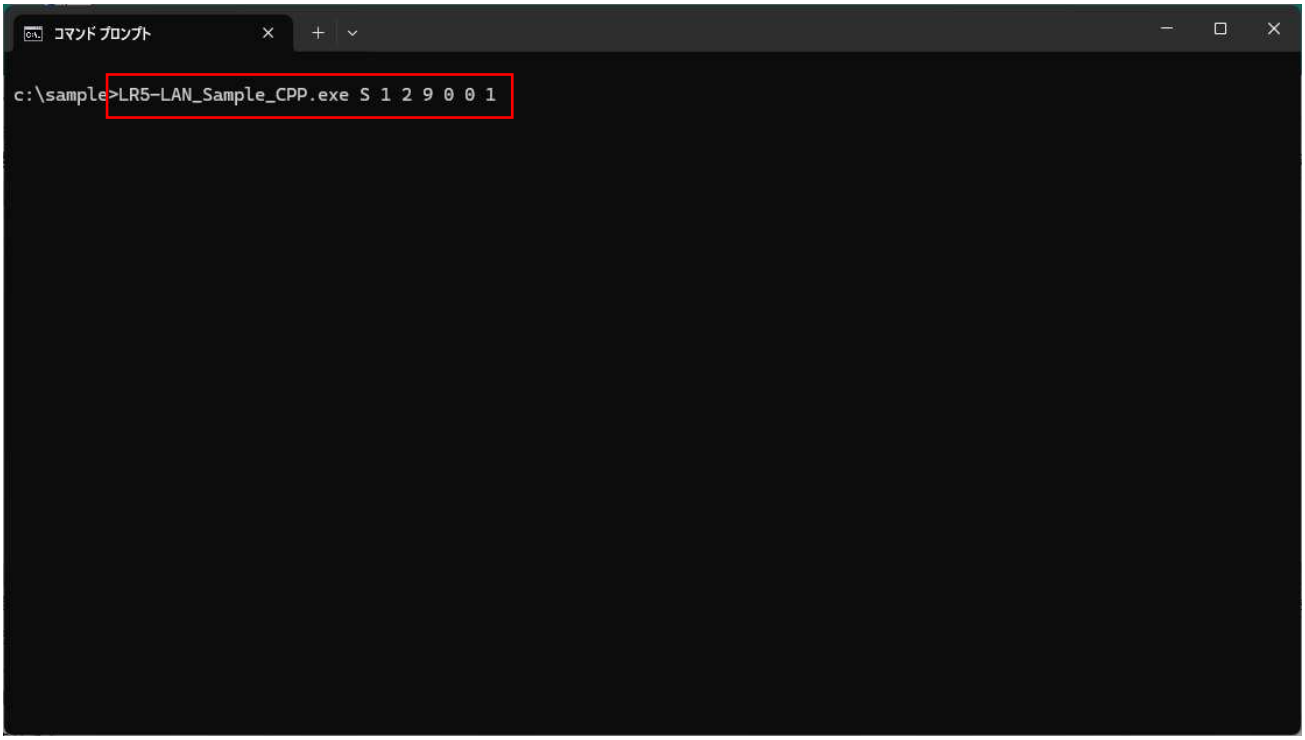
The development environment of the sample program is shown below.

Development Environment		Remarks
Development OS	Windows11 64bit	
Development Language	C++	
Application	CUI APPLICATION	
Development tool	VisualStudio2022 Professional	

3. Application Overview

3.1. Command Operation

Open Command Prompt, navigate to where the LR5-LAN_Sample_CPP.exe created during the build is located and specify the command line arguments to execute commands for each operation.



3.1.1. Command list

command name	content
Operation control command	Control each color pattern and buzzer (On/Off) of the LED unit
Clear Command	Turn off the LED unit and turn off the buzzer
Status Acquisition Command	Status of signal lines/contact inputs and USED TO ACQUIRE STATUS OF SIGNAL LINES AND THE STATUS OF THE LED UNIT AND ALARM.

3.1.2. Operation control command

Execute command with the following command line arguments

No.	Command Line Argument	Value
1	Command ID	S
2	LED Unit Red	Off:0 On:1 Flashing(slow):2 Flashing(medium):3 Flashing(fast):4 Single flash:5 Double flash:6 Triple flash:7 No change:9
3	LED Unit Amber	
4	LED UNIT GREEN	
5	LED Unit Blue	
6	LED UNIT WHITE	
7	Alarm Pattern	Off:0 On:1 No change:9

e.g.):LR5-LAN _Sample_CPP.exe S 1 2 9 0 0 1

3.1.3. Clear Command

Execute command with the following command line arguments

No.	Command Line Argument	Value
1	Command ID	C

e.g.):LR5-LAN _Sample_CPP.exe C

3.1.4. Status Acquisition Command

Execute command with the following command line arguments

No.	Command Line Argument	Value
1	Command ID	G

e.g.):LR5-LAN Sample_CPP.exe G

3.2. Function Description

3.2.1. Function List

Function Name	Explanation
SocketOpen	Connect to LR5-LAN
SocketClose	close the socket
SendCommand	send command
PNS_RunControlCommand	SEND PNS COMMAND OPERATION CONTROL COMMANDS
PNS_ClearCommand	SEND CLEAR PNS COMMAND
PNS_GetDataCommand	Send PNS Command Status Acquisition Command

3.2.2. Connect to LR5-LAN

Function Name	int SocketOpen(std::string ip, int port)	
Parameters	std::string ip	LR5-LAN IP address
	int port	LR5-LAN port number
Return Value	int	Success: 0, Failure: other than 0
Explanation	Connect to LR5-LAN with specified IP address and port number using socket communication	

How to use functions	// Definition of Socket class variables
	SOCKET sock = NULL;
	
	// Main function
	int main(int argc, char* argv[])
	{
	  // Connect to LR5-LAN
	  ret = SocketOpen("192.168.10.1", 10000);
	  if (ret == -1)
	  {
	  return;
	  }
	}	

Remarks	Please refer to 「4.1Connect to LR5-LAN」For The Program Overview.	

3.2.3. close socket

Function Name	void SocketClose()
Parameters	None
Return Value	None
Explanation	CLOSE THE SOCKET CONNECTED TO LR5-LAN

How to use functions	// Main function
	int main(int argc, char* argv[])
	{
	  // Connect to LR5-LAN
	  ret = SocketOpen("192.168.10.1", 10000);
	  if (ret == -1) {
	  return;
	  }
	
	  // close socket
	  SocketClose();
	}

Remarks	Please refer to 「4.2close socket」For The Program Overview.

3.2.4. Send Command

Function Name	int SendCommand(char* sendData, int sendLength, char* recvData, int recvLength)	
Parameters	char* sendData	Transmission Data
	int sendLength	Transmission Data Size
	char* recvData	Received Data
	int recvLength	Received Data Size
Return Value	int	Success: 0, Failure: other than 0
Explanation	Send data to the connected LR5-LAN and return response data	

How to use functions	// Main function
	int main(int argc, char* argv[])
	{
	// Connect to LR5-LAN
	ret = SocketOpen("192.168.10.1", 10000);
	if (ret == -1) {
	return;
	}
	
	// Create transmission data
	char sendData[7];
	char recvData;
	sendData[0] = 0x41;
	sendData[1] = 0x42;
	sendData[2] = 0x53;
	sendData[3] = 0x00;
	sendData[4] = 0x00;
	sendData[5] = 0x00;
	sendData[6] = 0x01;
	
	// Send Command
	ret = SendCommand(sendData, PNS_COMMAND_HEADER_LENGTH + sizeof(groupNo), recvData, sizeof(recvData));
	if (ret != 0) {
	puts("failed to send data");
	return -1;
	}
	
	// close socket
	SocketClose();
	}	

Remarks	Please refer to 「4.3 Send Command」For The Program Overview.	

3.2.5. PNS Command Operation Control Command Transmission

Function Name	int PNS_RunControlCommand(PNS_RUN_CONTROL_DATA runControlData)	
Parameters	PNS_RUN_CONTROL_DATA runControlData	TRANSMISSION DATA THAT CONTROLS EACH COLOR PATTERN AND BUZZER OF THE LED UNIT For Details, See「3.4.1Motion control data structure」For The Program Overview.
Return Value	Int	Success: 0, Failure: other than 0
Explanation	SEND PNS COMMAND OPERATION CONTROL COMMANDS TO CONTROL EACH COLOR PATTERN AND BUZZER OF THE LED UNIT	

How to use functions	// Main function
	int main(int argc, char* argv[])
	{
	// Connect to LR5-LAN
	ret = SocketOpen("192.168.10.1", 10000);
	if (ret == -1) {
	return;
	}
	// PNS Command Operation Control Command Transmission
	// Led pattern0：Off
	// Led pattern1：On
	// Led pattern2：Flashing(slow)
	// Led pattern3：Flashing(medium)
	// Led pattern4：Flashing(fast)
	// Led pattern5：Single flash
	// Led pattern6：Double flash
	// Led pattern7：Triple flash
	// Led pattern9：No change
	// Alarm Pattern0：Off
	// Alarm Pattern1：On
	// Alarm Pattern9：No change
	PNS_RUN_CONTROL_DATA runControlData;
	runControlData.ledRedPattern = PNS_RUN_CONTROL_LED_ON;
	runControlData. ledAmberPattern= PNS_RUN_CONTROL_LED_BLINKING_SLOW;
	runControlData. ledGreenPattern= PNS_RUN_CONTROL_LED_NO_CHANGE;
	runControlData. ledBluePattern= PNS_RUN_CONTROL_LED_OFF;
	runControlData. ledWhitePattern= PNS_RUN_CONTROL_LED_FLASHING_TRIPLE;
	runControlDatabuzzerPattern = PNS_RUN_CONTROL_BUZZER_RING;
	PNS_RunControlCommand(runControlData);
	
	// close socket
	SocketClose();
	}	

Remarks	Please refer to「3.2.5.PNS Command Operation Control Command Transmission」For The Program Overview.	

3.2.6. Send Clear Command For PNS Command

Function Name	int PNS_ClearCommand()	
Parameters	None	
Return Value	Int	Success: 0, Failure: other than 0
Explanation	SEND THE PNS CLEAR COMMAND TO TURN OFF THE LED UNIT AND STOP THE BUZZER	

How to use functions	// Main function
	int main(int argc, char* argv[])
	{
	// Connect to LR5-LAN
	ret = SocketOpen("192.168.10.1", 10000);
	if (ret == -1) {
	return;
	}
	
	// Send Clear Command For PNS Command
	PNS_ClearCommand();
	
	// close socket
	SocketClose();
	}	

Remarks	Please refer to「4.5Send Clear Command For PNS Command」For The Program Overview.	

3.2.7. Send PNS Command Status Acquisition Command

Function Name	int PNS_GetDataCommand(PNS_STATUS_DATA* statusData)	
Parameters	PNS_STATUS_DATA* statusData	Status Acquisition Command の Received Data(LED UNIT AND BUZZER STATUS) For Details, See 「3.4.2Operation control status data」For The Program Overview.
Return Value	Int	Success: 0, Failure: other than 0
Explanation	Send the status acquisition command of the PNS command to acquire the status of the led unit and buzzer..	

How to use functions	// Main function
	int main(int argc, char* argv[])
	{
	// Connect to LR5-LAN
	ret = SocketOpen("192.168.10.1", 10000);
	if (ret == -1) {
	return;
	}
	
	// Send PNS Command Status Acquisition Command
	PNS_STATUS_DATA statusData;
	PNS_GetDataCommand(&tatusData);
	
	// close socket
	SocketClose();
	}	

Remarks	Please refer to「4.6Send PNS Command Status Acquisition Command」For The Program Overview.	

3.3. Constant Description

3.3.1. Product Differentiation

Constant name	Value	Explanation
PNS_PRODUCT_ID	0x4142	LR5-LAN PRODUCT CLASSIFICATION

3.3.2. PNS Command Identifier

Constant name	Value	Explanation
PNS_RUN_CONTROL_COMMAND	0x53	Operation control command
PNS_CLEAR_COMMAND	0x43	Clear Command
PNS_GET_DATA_COMMAND	0x47	Status Acquisition Command

3.3.3. PNS Command Send Data Buffer Size

Constant name	Value	Explanation
PNS_COMMAND_HEADER_LENGTH	6	Product Differentiation to Data Size Buffer Size

3.3.4. PNS Command Response Data

Constant name	Value	Explanation
PNS_ACK	0x06	Normal Response
PNS_NAK	0x15	Abnormal Response

3.3.5. LED unit pattern for operation control commands

Constant name	Value	Explanation
PNS_RUN_CONTROL_LED_OFF	0x00	Off
PNS_RUN_CONTROL_LED_ON	0x01	On
PNS_RUN_CONTROL_LED_BLINKING_SLOW	0x02	Flashing(slow)
PNS_RUN_CONTROL_LED_BLINKING_MEDIUM	0x03	Flashing(medium)
PNS_RUN_CONTROL_LED_BLINKING_HIGH	0x04	Flashing(fast)
PNS_RUN_CONTROL_LED_FLASHING_SINGLE	0x05	Single flash
PNS_RUN_CONTROL_LED_FLASHING_DOUBLE	0x06	Double flash
PNS_RUN_CONTROL_LED_FLASHING_TRIPLE	0x07	Triple flash
PNS_RUN_CONTROL_LED_NO_CHANGE	0x09	No change

3.3.6. Buzzer pattern for operation control commands

Constant name	Value	Explanation
PNS_RUN_CONTROL_BUZZER_STOP	0x00	Off
PNS_RUN_CONTROL_BUZZER_RING	0x01	On
PNS_RUN_CONTROL_BUZZER_NO_CHANGE	0x09	No change

3.4. Structure Description

3.4.1. Motion control data structure

名前	PNS_RUN_CONTROL_DATA

Definition	typedef struct
	{
	// LED Unit Red pattern
	unsigned char ledRedPattern;
	// LED Unit Amber pattern
	unsigned char ledAmberPattern;
	// LED UNIT GREEN pattern
	unsigned char ledGreenPattern;
	// LED Unit Blue pattern
	unsigned char ledBluePattern;
	// LED UNIT WHITE pattern
	unsigned char ledWhitePattern;
	// Buzzer Status
	unsigned char buzzerMode;
	}PNS_RUN_CONTROL_DATA;

Explanation	Structure of each pattern and buzzer status of the LED unit in the Data Area sent by the Operation control command

3.4.2. Operation control status data

名前	PNS_STATUS_DATA

Definition	typedef struct
	{
	// Led pattern1～5
	unsigned char Led pattern[5];
	// Buzzer Mode
	unsigned char buzzer;
	}PNS_STATUS_DATA;

Explanation	Operation control Status Acquisition Command response data LED UNIT AND BUZZER STATUS structure

4. Program Overview

Describe only the main points of the program's operation.。

4.1. Connect to LR5-LAN

Program	Explanation

main.cpp
`WSADATA wsaData;`
`SOCKET sock = NULL;`
	→Define member variables of the socket information structure
	→Definition of socket member variables

main.cpp SocketOpen()
`int SocketOpen(std::string ip, int port)`
`{`
`// Initialize winsock2`
`if (WSAStartup(MAKEWORD(2, 0), &wsaData))`
`{`
`std::cout << "reset winsock failed" << std::endl;`
`return -1;`
`}`
`// Create a socket`
`sock = socket(AF_INET, SOCK_STREAM, 0);`
`if (sock == INVALID_SOCKET)`
`{`
`std::cout << "make socket failed" << std::endl;`
`return -1;`
`}`
`// Set the IP address and port`
`struct sockaddr_in addr;`
`addr.sin_family = AF_INET;`
`addr.sin_port = htons(port);`
`addr.sin_addr.S_un.S_addr = inet_addr(ip.c_str());`
`// Connect to LA-POE`
`if (connect(sock, (struct sockaddr*)&addr, sizeof(addr)`
`{`
`std::cout << "connect failed" << std::endl;`
`return -1;`
`}`
`return 0;`
`}`
	→Initializing winsock2
	→Create a socket instance
	→Specify the device IP address and port number
	Default IP address: 192.168.10.1
	Default port number: 10000
	→Connect to the device using the socket Connect function

4.2. close socket

Program	Explanation

main.cpp SocketClose()
`void SocketClose()`
`{`
`  // Close the socket.`
`  closesocket(sock);`
 	
`  // Exit process of winsock2`
`  WSACleanup();`
`}`
	→close socket
	
	→Winsock2 termination process

4.3. Send Command

Create transmission data in the transmission data format for each command and send the command data to LR5-LAN Please refer to「3.2.5.PNS Command Operation Control Command Transmission」and onwards for the transmission data format of each command.

Program	Explanation

main.cpp SendCommand()
`int ret;`
 	
`if (sock == NULL)`
`{`
`  std::cout << "socket is not" << std::endl;`
`  return -1;`
`}`
 	
`// Send`
`ret = send(sock, sendData, sendLength, 0);`
`if (ret < 0)`
`{`
`  std::cout << "failed to send" << std::endl;`
`  return -1;`
`}`
 	
`// Receive response data`
`std::memset(recvData, 0, recvLength);`
`ret = recv(sock, recvData, recvLength, 0);`
`if (ret < 0)`
`{`
`  std::cout << "failed to recv" << std::endl;`
`  return -1;`
`}`
	→Send the created Transmission Data using the Send function
	
	→After sending, use the recv function to get a response from the device.

4.4. PNS Command Operation Control Command Transmission

Program	Explanation

main.cpp PNS_RunControlCommand()
`int ret;`
`char sendData[PNS_COMMAND_HEADER_LENGTH + sizeof(runControl`
`char recvData[1];`
`std::memset(sendData, 0, sizeof(sendData));`
`std::memset(recvData, 0, sizeof(recvData));`
 	
`// Product Category (AB)`
`sendData[0] = PNS_PRODUCT_ID >> 8;`
`sendData[1] = (char)(PNS_PRODUCT_ID | 0xFF00);`
 	
`// Command identifier (S)`
`sendData[2] = PNS_RUN_CONTROL_COMMAND;`
 	
`// Empty (0)`
`sendData[3] = 0;`
 	
`// Data size`
`sendData[4] = sizeof(runControlData) >> 8;`
`sendData[5] = (char)(sizeof(runControlData) | 0xFF00);`
 	
`// Data area`
`std::memcpy(&sendData[6], &runControlData, sizeof(runContrc`
 	
`// Send PNS command`
`ret = SendCommand(sendData, PNS_COMMAND_HEADER_LENGTH + siz`
`if (ret != 0)`
`{`
`  std::cout << "failed to send data" << std::endl;`
`  return -1;`
`}`
 	
`// check the response data`
`if (recvData[0] == PNS_NAK)`
`{`
`  // receive abnormal response`
`  std::cout << "negative acknowledge" << std::endl;`
`  return -1;`
`}`
	Create Transmission Data in the following order
	→1st byte：Product Differentiation（A：0x41）
	→：Product Differentiation（B：0x42）
	→3rd byte：ID（S：0x53）
	→4th byte：Unused（0x00）
	→5th byte：Data Size（0x00）
	→6th byte：Data Size（0x06）
	→7～1：Data Area
	Data size is 6 bytes
	Set the value of "3.4.1 Motion control data structure" in the Data Area.
	
	→Call "4.3 Send Command/Receive" and send data to the device
	
	→Check response data after sending
	Normal Response：ACK（0x06）
	Abnormal Response：NAK（0x15）

4.5. Send Clear Command For PNS Command

Program	Explanation

main.cpp PNS_ClearCommand()
`int ret;`
`char sendData[PNS_COMMAND_HEADER_LENGTH];`
`char recvData[1];`
`std::memset(sendData, 0, sizeof(sendData));`
`std::memset(recvData, 0, sizeof(recvData));`
 	
`// Product Category (AB)`
`sendData[0] = PNS_PRODUCT_ID >> 8;`
`sendData[1] = (char)(PNS_PRODUCT_ID | 0xFF00);`
 	
`// Command identifier (C)`
`sendData[2] = PNS_CLEAR_COMMAND;`
 	
`// Empty (0)`
`sendData[3] = 0;`
 	
`// Data size`
`sendData[4] = 0;`
`sendData[5] = 0;`
 	
`// Send PNS command`
`ret = SendCommand(sendData, PNS_COMMAND_HEADER_LENGTH, recv`
`if (ret != 0)`
`{`
`  std::cout << "failed to send data" << std::endl;`
`  return -1;`
`}`
 	
`// check the response data`
`if (recvData[0] == PNS_NAK)`
`{`
`  // receive abnormal response`
`  std::cout << "negative acknowledge" << std::endl;`
`  return -1;`
`}`
	Create Transmission Data in the following order
	→1st byte：Product Differentiation（A：0x41）
	→：Product Differentiation（B：0x42）
	→3rd byte：ID（C：0x43）
	→4th byte：Unused（0x00）
	→5th byte：Data Size（0x00）
	→6th byte：Data Size（0x00）
	Data size is 0 bytes
	No data area
	
	→Call “4.3 Send Command/Receive” and send data to the device
	
	→Check response data after sending
	Normal Response：ACK（0x06）
	Abnormal Response：NAK（0x15）

4.6. Send PNS Command Status Acquisition Command

Program	Explanation

main.cpp PNS_GetDataCommand()
`int PNS_GetDataCommand(PNS_STATUS_DATA* statusData)`
`{`
`  int ret;`
`  char sendData[PNS_COMMAND_HEADER_LENGTH];`
`  char recvData[sizeof(PNS_STATUS_DATA)];`
`  std::memset(sendData, 0, sizeof(sendData));`
`  std::memset(recvData, 0, sizeof(recvData));`
`  std::memset(statusData, 0, sizeof(PNS_STATUS_DATA));`
 	
`  // Product Category (AB)`
`  sendData[0] = PNS_PRODUCT_ID >> 8;`
`  sendData[1] = (char)(PNS_PRODUCT_ID | 0xFF00);`
 	
`  // Command identifier (G)`
`  sendData[2] = PNS_GET_DATA_COMMAND;`
 	
`  // Empty (0)`
`  sendData[3] = 0;`
 	
`  // Data size`
`  sendData[4] = 0;`
`  sendData[5] = 0;`
 	
`  // Send PNS command`
`  ret = SendCommand(sendData, PNS_COMMAND_HEADER_LENGTH, recvData`
`  if (ret != 0)`
`  {`
`  std::cout << "failed to send data" << std::endl;`
`  return -1;`
`  }`
 	
`  // check the response data`
`  if (recvData[0] == PNS_NAK)`
`  {`
`  // receive abnormal response`
`  std::cout << "negative acknowledge" << std::endl;`
`  return -1;`
`  }`
 	
`  // LED unit R pattern 1 to 5`
`  std::memcpy(statusData->ledPattern, &recvData[0], sizeof(status`
 	
`  // Buzzer Mode`
`  statusData->buzzer = recvData[5];`
 	
`  return 0;`
`}`
	Create Transmission Data in the following order
	→1st byte：Product Differentiation（A：0x41）
	→：Product Differentiation（B：0x42）
	→3rd byte：ID（G：0x47）
	→4th byte：Unused（0x00）
	→5th byte：Data Size（0x00）
	→6th byte：Data Size（0x00）
	Data size is 0 bytes
	No data area
	
	→Call “4.3 Send Command/Receive” and send data to the device
	
	→Check response data after sending
	Normal Response: The response data in “3.4.2 Operation control status data” is obtained.
	Abnormal Response：NAK（0x15）
	
	Acquire each data of response data using the following process.
	→LED UNIT STATUS
	・1st byte：LED Unit Redstatus
	・：LED Unit Amberstatus
	・3rd byte：LED UNIT GREENstatus
	・4th byte：LED Unit Bluestatus
	・5th byte：LED UNIT WHITEstatus
	・6th byte：Buzzer status