БГУИР

Кафедра ЭВМ

Лабораторная работа №2

Пакетная передача данных

Выполнил студент группы 950503 Проверил ассистент кафедры ЭВМ

Сякачёв П.В. Марцинкевич В.А.

Минск 2021

Для именования порции информации, передаваемой по каналам компьютерных (и не только компьютерных) сетей, используется обобщенный термин пакет (packet). Пакет содержит последовательно сформированные станцией-передатчиком поля (fields), предназначенные для их интерпретации в станции-приемнике. В общем случае, пакеты могут быть самыми разнообразными (как по структуре, так и по длине).

Типовая структура пакета включает в себя:

Флаг начала пакета – позволяет определить начало пакета.

Адрес назначения – позволяет указать станцию, для которой предназначен пакет.

Адрес источника – позволяет указать станцию, из которой был отправлен пакет.

Прочие поля – специфические поля определенной реализации.

Данные – полезная нагрузка пакета.

Контрольная сумма – позволяет проверить был ли искажен пакет при передаче (контрольная сумма не всегда может определить искажение пакета, например бит четности ошибочно примет двойную ошибку за корректные данные).

Часть пакета, которая включает поля, расположенные до начала данных, называют заголовком, а поля, расположенные после данных хвостовиком.

При продвижении информации между уровнями возникает необходимость в преобразованиях структур данных. Преобразования выражаются в инкапсуляции и декапсуляции. Под инкапсуляцией понимается вкладывание пакета определенного вышестоящего уровня в поле данных пакета смежного нижестоящего уровня в процессе подготовки к передаче, то есть при продвижении сверху вниз. Под декапсуляцией понимается обратное действие после приема, то есть при продвижении снизу вверх.

Для правильно интерпретации пакета его необходимо полностью считать из канала, с соблюдением порядка чтения пакета. Так как станция-приемник может подключиться к каналу в произвольный момент времени, возникает проблема с определение флага начала пакета. Так как флаг начала пакета является зарезервированной последовательностью, такая же последовательность может появиться в пакете в качестве информации, передаваемой в полезной нагрузке пакета.

Для решения данной проблемы используются действия, изменяющие следующую за флагом цифровую последовательность. Они называются битстаффинг и байтстаффинг в бит-ориентированных и байт-ориентированных системах соответственно.

При бит-стаффинге совпадающая с начальным флагом последовательность разбивается при помощи вставки бита с соответствующим значением. Классическим флагом начала пакета является флаг со значение 01111110. Оптимальная реализация битстаффинга заключается в вставке после нуля и шести единиц седьмой единицы на передатчике, а на приемнике ее удаление. Обычно реализуется аппаратно.

При байт-стаффинге происходит манипуляция байтами, а не битами. Это облегчает работу при программировании (позволяет избегать битовых операций), но является более затратным. Решением проблемы при байт-стаффинге является использование ESC-символов, которые сообщают о факте замены.

int Server()

{

HANDLE readEnd = CreateEvent(NULL, FALSE, FALSE, L"readEnd");

HANDLE writeEnd = CreateEvent(NULL, FALSE, FALSE, L"writeEnd");

HANDLE hExit = CreateEvent(NULL, FALSE, FALSE, L"exit");

HANDLE hWork = CreateEvent(NULL, FALSE, FALSE, L"work");

HANDLE hFile = CreateFile(L"COM1", GENERIC\_WRITE | GENERIC\_READ, 0, NULL, OPEN\_EXISTING, 0, NULL);

if (hFile == NULL)

{

printf("Error\n");

exit(GetLastError());

}

SetCommMask(hFile, EV\_RXCHAR);

SetupComm(hFile, 1500, 1500);

COMMTIMEOUTS CommTimeOuts;

CommTimeOuts.ReadIntervalTimeout = 0xFFFFFFFF;

CommTimeOuts.ReadTotalTimeoutMultiplier = 0;

CommTimeOuts.ReadTotalTimeoutConstant = 1200;

CommTimeOuts.WriteTotalTimeoutMultiplier = 0;

CommTimeOuts.WriteTotalTimeoutConstant = 1200;

if (!SetCommTimeouts(hFile, &CommTimeOuts))

{

CloseHandle(hFile);

hFile = INVALID\_HANDLE\_VALUE;

return 1;

}

DCB ComDCM;

memset(&ComDCM, 0, sizeof(ComDCM));

ComDCM.DCBlength = sizeof(DCB);

GetCommState(hFile, &ComDCM);

ComDCM.BaudRate = setBaudrate();

ComDCM.ByteSize = 8;

ComDCM.Parity = NOPARITY;

ComDCM.StopBits = ONESTOPBIT;

ComDCM.fAbortOnError = TRUE;

ComDCM.fDtrControl = DTR\_CONTROL\_DISABLE;

ComDCM.fRtsControl = RTS\_CONTROL\_DISABLE;

ComDCM.fBinary = TRUE;

ComDCM.fParity = FALSE;

ComDCM.fInX = FALSE;

ComDCM.fOutX = FALSE;

ComDCM.XonChar = 0;

ComDCM.XoffChar = (unsigned char)0xFF;

ComDCM.fErrorChar = FALSE;

ComDCM.fNull = FALSE;

ComDCM.fOutxCtsFlow = FALSE;

ComDCM.fOutxDsrFlow = FALSE;

ComDCM.XonLim = 128;

ComDCM.XoffLim = 128;

if (!SetCommState(hFile, &ComDCM))

{

CloseHandle(hFile);

hFile = INVALID\_HANDLE\_VALUE;

return 2;

}

PORT1(readEnd, writeEnd, hFile, hExit, hWork);

CloseHandle(hFile);

return 0;

}

Данный метод предназначен для инициализации COM порта при помощи структуры DCB.

В начале метода открывается порт для чтения/записи и инициализируются структуры для асинхронного чтения/записи. Далее происходит начальная настройка порта.

string encode(string buffer) {

string encodedData;

int i = 0;

while (i < buffer.size())

encodedData += bitset<8>(buffer[i++]).to\_string();

return encodedData;

}

string decode(string buffer) {

string decodedData;

for (int i = 0; i < buffer.size() / 8; i++)

{

decodedData += strtol(buffer.substr(i \* 8, 8).c\_str(), NULL, 2);

}

return decodedData;

}

Данные методы отвечают перевод строки в битовую последовательность (кодирование) и обратно (декодирование).

vector<string> packData(string buffer) {

vector<string> packs;

for (int i = 0; i <= buffer.size() / PACKET\_SIZE; i++)

{

packs.push\_back("");

string packedMsg(buffer.substr(i \* PACKET\_SIZE, PACKET\_SIZE));

int poz = 0;

while (1)

{

poz = packedMsg.find("111111", poz);

if (poz != -1)

{

packedMsg.insert(poz, "1");

poz += 6;

}

else break;

}

while (packedMsg.size() != PACKET\_SIZE + 8) packedMsg.push\_back('0');

int ones = 0;

for (auto digit : packedMsg) if (digit == '1') ones++;

packedMsg.insert(0, flag);

packs[i] = decode(packedMsg);

packs[i].insert(1, to\_string(ones % 2));

}

return packs;

}

Данный метод отвечает за разделение данных и упаковку их в пакеты. Тут же производится бит-стаффинг(если он требуется).

string unpackData(string packedData) {

int parityBit = (int)decode(packedData.substr(8, 8))[0] - 48;

packedData.erase(8, 8);

int poz = packedData.find(flag);

if (poz == -1) return "";

else packedData.erase(0, 8);

int ones = 0;

for (auto digit : packedData) if (digit == '1') ones++;

if (ones % 2 != parityBit) return "";

poz = 0;

while (1)

{

poz = packedData.find("1111111", poz);

if (poz != -1)

{

packedData.erase(poz, 1);

packedData.push\_back('0');

poz += 6;

}

else break;

}

return packedData;

}

Данный метод обратен предыдущему.

Вывод: в ходе выполнения данной лабораторной работы был получен опыт разделения данных на пакеты и работы с такими пакетами.