Telekomunikacja - laboratorium	Studia stacjonarne - inzynierskie

Nazwa zadania	Ćwiczenie nr 2. Protokół Xmodem				
Dzień	wtorek	Godzina	17:30	Rok akademicki	2019/2020

Imię i Nazwisko	Paweł Guzek (224304)
Imię i Nazwisko	Michał Maksajda (224369)
Imię i Nazwisko	-

Opis programu, rozwiązania problemu.

Celem zadania jest napisanie programu, który wykorzystuje komunikację poprzez port szeregowy RS-232. Program powinien działać w środowisku Windows, powinien obsługiwać dwukierunkową transmisję w oparciu o implementację protokołu XMODEM. Przesyłanie plików powinno działać dla Algebraicznej Sumy Kontrolnej (ASK) oraz cyklicznego kodu nadmiarowego (CRC).

Po uruchomieniu pliku wykonywalnego użytkownik wybiera z pośród czterech trybów pracy programu:

1) Nadawca (CRC)
2) Nadawca (NAK)
3) Odbiorca (CRC)
4) Odbiorca (NAK)

Po wybraniu trybu pracy programu użytkownik musi wybrać port COM na którym program włączy nasłuchiwanie, robi to wpisując odpowiedni numer portu COM na klawiaturze.

Następnie zostaniemy poproszeni o podanie ścieżki do pliku, którego chcemy użyć- w zależności od trybu Nadawca / Odbiorca plik zostanie przesłany lub zostaną do niego zapisane dane.

Po wykonaniu ostatniej czynności program rozpocznie procedurę uruchamiania nasłuchiwania na porcie wybranym przez użytkownika. W pierwszej kolejności zostanie utworzony uchwyt do portu który chcemy obsłużyć, uchwyt wymaga dostępu do zapisu oraz odczytu danych. Po zestawieniu połączenia z portem, pobieramy jego domyślne ustawienia oraz ustawiamy jednorodne wymagania do obsługi różnych klientów xmodem. Parametry wykorzystane do transmisji to:

- prędkość transmisji 9600 bps,

- brak bitu parzystości,

- jeden bit stopu,

- rozmiar bajtu dla wykorzystywanego systemu to 8 bitów.

Ponad to wyłączamy DTE(data-terminal-ready), RTS(request-to-send – ramka kontrolna MAC), monitorowanie CTS(clear-to-send), monitorowanie DST(data-set-ready), zgłaszanie błędów IO, transmisję znaków XON/XOFF oraz odrzucanie NULL-bajtów.

Kolejno zostają ustawione limity czasu oczekiwania na rozpoczęcie transmisji.

W zależności od wybranego trybu pracy odbiornik/nadajnik wykonywane jest:

1) Odbiornik

   Przesyłanie odpowiedniego bitu do portu w celu przekazania nadajnikowi gotowości do odbioru:

   a) Dla trybu CRC wysyłamy znak ‘C’ o wartości 0x43.
   b) Dla trybu ASK wysyłamy znak ‘NAK’ o wartości 0x15.

   Identyfikuje to rozmiar ramki której będziemy używać- dla ‘C’ wynosi on 133 bajty, natomiast dla ‘NAK’ wynosi 132 bajty.

   Gdy nadajnik otrzyma nasza informację powinien rozpocząć transmisję ramek z danymi. Każda ramka składa się z [1B typRamki, 1B numerPakietu, 1B dopelnieniePakietu, 128B dane, 2B lub 1B sumaKontrolna].

   Typ przesyłanej ramki jest oznaczany przez znak ‘SOH’ (Start of Header) posiada wartość 0x01.

   Gdy ramka zostanie wykryta oznacza to, że prawidłowo zestawiono połączenie. W następnych krokach dla każdej otrzymanej ramki będziemy sprawdzać jej typ:

   a) Ramka EOT (End of Transmission)- informuje nas o pomyślnym zakończeniu przesyłu danych,
   b) Ramka CAN (Cancel)- kończy przesył danych,
   c) Ramka NAK (Not Acknowledge)- informuje nadawcę o błędzie w przesłanej ramce (prosi o ponowne przeslanie)
   d) Ramka ACK (Acknowledge)- informuje nadawcę o poprawnym otrzymaniu ramki

   Poprawność ramki sprawdzamy w dwóch krokach:

   a) Suma numeru pakietu oraz jego dopełnienia jest wartością 255 (0xFF)
   b) Wyliczona suma kontrolna ze 128B danych jest równa przesłanej sumie kontrolnej.
      a. Algebraiczna Suma Kontrolna

         Wyliczana jako suma wszystkich 128B wartości z fragmentu dane przesłanej ramki modulo 256

b. CRC

Wyliczana dla wszystkich 128B wartości z fragmentu dane przesłanej ramki, Wykonujemy operację XOR na 8 najstarszych bitach sumy kontrolnej z poprzedniego kroku, następnie 8 razy przesuwamy bity w stronę najstarszego, w momencie gdy najstarszy bit ma stan wysoki, wykonujemy operację XOR (dzielenia) przez generator 0x1021 (wielomian x16 + x12 + x5 + 1).

W przypadku braku poprawności przesyłany jest sygnał NAK, natomiast przeciwnie wysyłamy sygnał ACK.

Po pomyślnym przesłaniu ramki (wysłaniu ACK) program zapisuje blok danych do pliku oraz odbiera kolejną ramkę.

Przed zakończeniem transmisji danych (przed sygnałem EOT), program usuwa z 128B danych dopełnienie wartością 0x1A (ASCII 26 / ^Z) określonej według specyfikacji protokołu xmodem jako jego padding, oraz zapisuje wynik do pliku.

Po otrzymaniu sygnału EOT wysyłamy jego potwierdzenie ACK co kończy transmisję.

2) Nadajnik

Wysyłanie danych działa w podobny sposób, w pierwszej kolejności następuje oczekiwanie na sygnał gotowości do odbioru C lub NAK w zależności od typu transmisji. Następnie przygotowywana jest ramka z danymi, składa się z:

a) Typ ramki- SOH

b) Numer przesyłanego pakietu inkrementowany modulo 256 z każdym poprawnie przesłanym,

c) Dopełnienie do przesłanego pakietu opisane jako 0xFF pomniejszone o numerPrzeslanegoPakietu,

d) 128B danych- dopełnione wartością 0x1A (ASCII 26 / ^Z)

e) 2B lub 1B sumy kontrolnej wyliczonej jak powyżej (pkt. 1.b)

W taki sposób przygotowana ramka jest wysyłana, po otrzymaniu potwierdzenia ACK wykonujemy powyższe operację dla kolejnych ramek; w przypadku NAK przesyłamy ponownie ramkę; natomiast w przypadku CAN zaprzestajemy transferu.

Po przesłaniu wszystkich danych wysyłamy sygnał EOT aby powiadomić odbiorcę o zakończeniu transmisji, następnie czekamy na potwierdzenie sygnału EOT przez odbiorcę oraz kończymy transmisję.

Najważniejsze elementy kodu programu z opisem.

include/defines.h

```
const BYTE SOH = 0x01;     // Start of Header
const BYTE EOT = 0x04;     // End of Transmission
const BYTE ACK = 0x06;     // Acknowledge
const BYTE NAK = 0x15;     // Not Acknowledge  -> Start Algebraic transfer
const BYTE ETB = 0x17;     // End of Transmission Block (Return to Amulet OS mode)
const BYTE CAN = 0x18;     // Cancel (Force receiver to start sending C's)
const BYTE C   = 0x43;     // ASCII 'C'        -> Start CRC transfer
```

Podstawowe sygnały oraz ich wartości używane w trakcie transmisji.

-=-=-=-=-=-=-=-

include/DataBlock.h

Format przechowujący strukturę ramki (bez typu pakietu, z dodatkowym polem określającym status CRC16 lub ASK).

```
    void streamData(ofstream &ofstr, bool isLastPacket) {
        int skipAfter = 0;
        // jezeli ostatni pakiet zlicz ile bajtow paddingu zostalo dodanych
        if (isLastPacket) {
            for (int i = sizeof(data) / sizeof(data[0]) - 1; i >= 0; i--) {
                if (data[i] == 0x1A) { // padding (26)
                    skipAfter++;
                }
            }
        }
        // i mniejsze od dlugosci przeslanych danych pomniejszonych o dlugosc padding
        for (int i = 0; i < (sizeof(data) / sizeof(data[0])) - skipAfter; i++) {
            ofstr << data[i];
        }
    }
```

Funkcja służąca do zapisu danych, do pliku oraz usuwania dopełnienia nakładanego do ostatniej ramki transmisji.

```
    // przelicza sume kontrolna CRC16 lub Algebraiczna Sume Kontrolna
    int calculateChecksum() {
        int count = sizeof(data) / sizeof(data[0]); // dlugosc tablicy dane
        BYTE *ptr = data; // wsk na pierwszy element tablicy

        if (isCRC16) {                              // jezeli CRC to:
            int crc = 0;
            while (count--) {
```

```
                crc ^= (*ptr++ << 8);               // XOR starszej czesci crc z nowa liczba przesunieta do starszych
bitow
                for (int i = 0; i < 8; i++) {
                    if (crc & 0x8000) {             // sprawdzenie czy najstarszy bit jest rowny 1, nie chcemy go stracic
                        crc = (crc << 1) ^ 0x1021;  // wyrzucenie MSB oraz dodanie 0 jako LSB oraz XOR z generatorem
(Most/Lease Significant Bit)
                    } else {
                        crc <<= 1;                  // wyrzucenie MSB oraz dodanie 0 jako LSB
                    }
                }
            }
            return (crc & 0xFFFF);
        } else {                                    // jezeli ASK to:
            int checksum = 0;

            while (count--) {
                checksum += (*ptr++);               // sumuj wszystkie wartosci z dane
            }
            checksum %= 256;                        // modulo 256
            return checksum;
        }
    }



    int calculateBlockCRC() {
        // stworzenie liczby 16 bitowej z dwóch blokow 8 bit jako CRC[0]_CRC[1]
        return isCRC16 ? (CRC[0] << 8) | CRC[1] & 0x00FF : CRC[0];
    }


-=-=-=-=-=-=-=-

src/PortHandler.cpp

    portSettings.BaudRate = CBR_9600;                       // predkosc transmisji 9600 bps
    portSettings.Parity = NOPARITY;                         // brak bitu parzystosci
    portSettings.StopBits = ONESTOPBIT;                     // ustawienie bitu stopu (jeden bit)
    portSettings.ByteSize = 8;                              // liczba wysylanych bitow

Ustawienia połączenia


void PortHandler::receive(const string &fileName) {
    BYTE type = 0;
    for (int i = 0;; i++) {
        // Wysylamy sygnal gotowosci do odbioru
        WriteFile(portHandle, &transmissionType, 1, &bitsLengthInChar, nullptr);
        cout << "Oczekiwanie na SOH" << '\n';
// pobranie 1 bajtu z zasobu
        ReadFile(portHandle, &type, 1, &bitsLengthInChar, nullptr);
        cout << "Otrzymano odpowiedz: " << ((int) type) << '\n';
        if (SOH == type) {
// jezeli bajt SOH to rozpoczynamy transmisje
            cout << "Ustalono polacznie z nadawca." << '\n';
            break;
        }
        if (i == 100) {
            cout << "Nie ustanowono polacznia. Force EOT/" << '\n';
            exit(1);
        }
    }

    ofstream fout(fileName, ios::binary);
    cout << "Utworzono plik do odbioru, odczytywanie wiadomosci." << '\n';


    // Dopuki nie ma konca transmisji lub nie jest przerwana
    while (!(type == EOT || type == CAN)) {
        DataBlock dataBlock{};

        //odbieramy dane o rozmiarze pakietu bez pierwszego bajtu i wczytujemy je do niej
        ReadFile(portHandle, &dataBlock, numberOfBytesToRead - 1, &dataBlockSize, nullptr);

        dataBlock.print();
        //czy bedziemy korzystac z CRC-16 (tryb pracy C || NAK)
        dataBlock.isCRC16 = (transmissionType == C);

        do {
            // Sprawdzamy poprawnosc- przy bledzie wysylamy NAK, gdy jest OK ACK
            if (!dataBlock.isCorrect()) {
```

```
                WriteFile(portHandle, &NAK, 1, &bitsLengthInChar, nullptr);
            } else {
                WriteFile(portHandle, &ACK, 1, &bitsLengthInChar, nullptr);
            }

            //odbieramy kolejny bajt naglowka
            ReadFile(portHandle, &type, 1, &bitsLengthInChar, nullptr);
        } while (0 == type); //w przypadku zgubienia ACK/NAK wyslij jeszcze raz

        if (dataBlock.isCorrect()) {
            dataBlock.streamData(fout, EOT == type || CAN == type);
        }
    }

    if (CAN == type) {
        cout << "Wystapil blad przy przesylaniu" << '\n';
    }
    cout << "EOT FIN" << '\n';

    //potwierdzamy EOT
    WriteFile(portHandle, &ACK, 1, &bitsLengthInChar, nullptr);

    //sprawdzamy i potwierdzamy ETB
    ReadFile(portHandle, &type, 1, &bitsLengthInChar, nullptr);
    if (ETB == type) {
        cout << "ETB FIN" << '\n';
        WriteFile(portHandle, &ACK, 1, &bitsLengthInChar, nullptr);
    }

    CloseHandle(portHandle);
    fout.close();
}
```

Funkcja służąca do odbioru danych.

```
void PortHandler::send(const string &fileName) {
    BYTE type = 0;

    for (int i = 0; ; i++) {
        cout << "Oczekiwanie na C/NAK" << '\n';
        ReadFile(portHandle, &type, 1, &bitsLengthInChar, nullptr);    // oczekiwanie na probe nadania danych
        cout << "Otrzymano odpowiedz: " << ((int) type) << '\n';
        if (C == type || NAK == type) {                                // jezeli C lub ASK rozpoczynamy transmisje
            cout << "Ustalono polacznie z nadawca." << '\n';
            break;
        }
        if(i==100) {
            cout << "Nie ustanowono polacznia. Force EOT/" << '\n';
            exit(1);
        }
    }

    BYTE transmissionMethod = type;                          // zapisujemy metode transmisji C | ASK
    BYTE transmissionLength = C == type ? 133 : 132;         // ustalamy ilosc bajtow dla typu transmisji
    ifstream is(fileName, ios::binary);
    deque<unsigned char> buffStream(istreambuf_iterator<char>(is), {});    // wczytujemy wartosci ze strumienie do deque
    is.close();

    BYTE packetNo = 1;
    int globalPacket = 0;
    while(!buffStream.empty()) {                                   // jezeli mamy dane na liscie
        cout << "-=-=-=-=-=-=-=-=-=-=- " << globalPacket << "-=-=-=-=-=-=-=-=-=-=-" << '\n';
        cout << "Rozpoczynam komponowanie pakietu" << '\n';
        BYTE tmpType = 0;
        DataBlock dataBlock{};
        dataBlock.packetNoQue = packetNo;                          // ustawiamy numer pakietu
        dataBlock.packetNoNeg = 0xFF - packetNo;                   // wyliczamy dopelnienie
        dataBlock.isCRC16 = (C == transmissionMethod);             // czy otrzymano C

        for(int i=0; i< (sizeof(dataBlock.data)/ sizeof(dataBlock.data[0])); i++) {
            if(buffStream.empty()) {                   // jezeli lista jest pusta dopelnij blok wartosciami 26 (0x1A)
                dataBlock.data[i] = 26;
            } else {
                dataBlock.data[i] = buffStream.front(); // odczytaj wartosc z poczatka listy i przypisz do bloku danych
                buffStream.pop_front();                // usun wartosc z poczatku listy
            }
        }
        dataBlock.generateCRC();                       // generuj CRC dla bloku danych
```

```
        cout << "Wygenerowano dane pakietu" << '\n';

        do {
            // wyslij naglowek oraz blok danych
            WriteFile(portHandle, &SOH, 1, &bitsLengthInChar, nullptr);
            cout << "Przeslano naglowek SOH pakietu" << '\n';
            WriteFile(portHandle, &dataBlock, transmissionLength - 1, &bitsLengthInChar, nullptr);
            cout << "Przeslano dane pakietu" << '\n';

            // odczytaj informacje zwrotna
            ReadFile(portHandle, &tmpType, 1, &bitsLengthInChar, nullptr);
            cout << "Odczytano wartosc tmpType: " << (int) tmpType << '\n';

            if (ACK == tmpType) {
                cout << "Wyslano pakiet" << '\n';
                break;
            } else if (NAK == tmpType) {
                cout << "Blad wysylania pakietu, ponawianie" << '\n';
                continue;
            } else if (CAN == tmpType) {
                cout << "Polaczenie przerwane" << '\n';
                exit(1);
            }

        } while (0 != tmpType);

        packetNo++;
        globalPacket++;
    }

    cout << '\n' << "Przeslano wszystkie dane; przesylanie EOT" << '\n';
    do {
        WriteFile(portHandle, &EOT, 1, &bitsLengthInChar, nullptr);
        ReadFile(portHandle, &type, 1, &bitsLengthInChar, nullptr);
        if (ACK == type) {
            cout << "Potwierdzono zakonczenie transmisji" << '\n';
        } else {
            cout << "Wystapil blad przy konczeniu transmisji, ponawianie" << '\n';
        }
    } while (0 == type);

    CloseHandle(portHandle);
}
```

Funkcja do wysyłania danych.

Podsumowanie wnioski.

Xmodem jest bardzo prostym protokołem terminalowego przesyłania plików który nie nakłada dużego narzutu na transmisję oraz przetworzenie danych, może zostać zaimplementowany nawet na najprostszych urządzeniach nie będących w stanie obsłużyć bardziej zaawansowanych technik przesyłu danych. Ze względu na swoją prostotę można go łato zaimplementować samemu co stanowi doskonały fundament pod analizę kolejnych protokołów transmisji danych.