Symulacja Cyfrowa – Projekt

Symulacja sieci bezprzewodowej metodą interakcji procesów.

Marcin Przepiórkowski

Spis Treści

2.Treść zadania.

W sieci bezprzewodowej stacje nadawcze konkurują o dostęp do łącza. W losowych odstępach czasu CGP_k k-ta stacja nadawcza generuje pakiety gotowe do wysłania. Po uzyskaniu dostępu do łącza zgodnie z algorytmem A, k-ty terminal podejmuje próbę transmisji najstarszego pakietu ze swojego bufora. Czas transmisji wiadomości z k-tej stacji nadawczej do k-tej stacji odbiorczej wynosi CTP_k. Jeśli transmisja pakietu zakończyła się sukcesem, stacja odbiorcza przesyła potwierdzenie ACK (ang. Acknowledgment) poprawnego odebrania wiadomości. Czas transmisji ACK wynosi CTIZ. Jeśli transmisja pakietu nie powiodła się, stacja odbiorcza nie przesyła ACK. Odbiór pakietu uznajemy za niepoprawny, jeśli w kanale transmisyjnym wystąpiła kolizja. Przez kolizję rozumiemy nałożenie się jakiejkolwiek części jednego pakietu na inny pakiet (pochodzący z innego nadajnika). Brak wiadomości ACK po czasie (CTPk+ CTIZ) od wysłania pakietu jest dla stacji nadawczej sygnałem o konieczności retransmisji pakietu. Każdy pakiet może być retransmitowany maksymalnie LR razy. Dostęp do łącza w przypadku retransmisji opiera się na tych samych zasadach co transmisja pierwotna. Jeśli mimo LR-krotnej próby retransmisji pakietu nie udało się poprawnie odebrać, wówczas stacja nadawcza odrzuca pakiet i – jeśli jej bufor nie jest pusty – przystępuje do próby transmisji kolejnego pakietu. Opracuj symulator sieci bezprzewodowej zgodnie z metodą interakcji procesów.

Za pomocą symulacji wyznacz:

Wartość parametru L, która zapewni średnią pakietową stopę błędów (uśrednioną po K odbiornikach) nie większą niż 0.1, a następnie:

- średnią liczbę retransmisji pakietów,

- przepływność systemu mierzoną liczbą poprawnie odebranych pakietów w jednostce czasu,

- średnie opóźnienie pakietu, tzn. czas jaki upływa między pojawieniem się pakietu w buforze, a jego poprawnym odebraniem,

- średni czas oczekiwania, tzn. czas między pojawieniem się pakietu w buforze, a jego opuszczeniem

Sporządź wykres zależności przepływności systemu oraz średniej i maksymalnej pakietowej stopy błędów w zależności od wartości L.

Parametry

CGP_k - zmienna losowa o rozkładzie wykładniczym o intensywności L. Wylosowany czas należy zaokrąglić do części dziesiętnej milisekundy.

CTP_k – zmienna losowa o rozkładzie jednostajnym w przedziale {1, 2, …, 10} ms

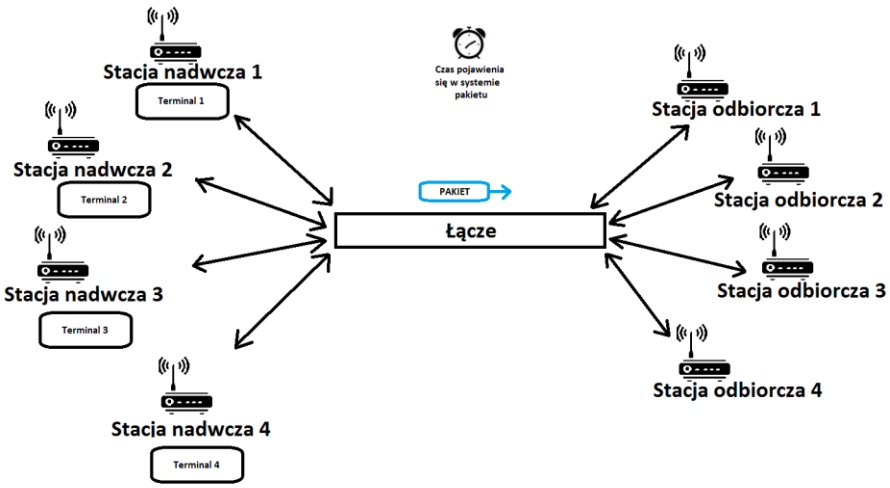
R – zmienna losowa o rozkładzie jednostajnym w przedziale <0, $(2^r -1)$>

CTIZ = 1 ms

CSC = 1 ms

3. Krótki opis modelu symulacyjnego:

a. schemat modelu symulacyjnego



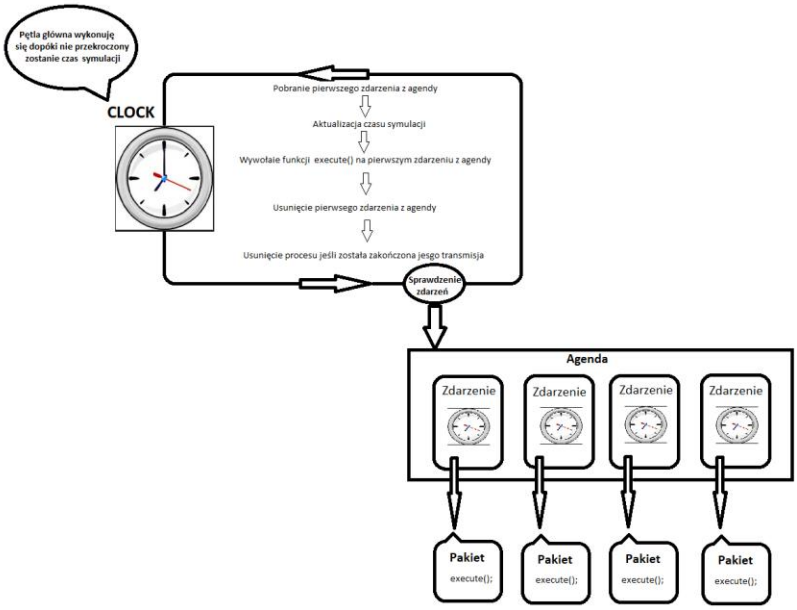
Rys.1. Schemat modelu symulacyjnego.

b. Opis klas wchodzących w skład systemu i ich atrybutów

Obiekt	**Nazwa klasy implementującej obiekt**	**Opis**	**Atrybuty**
Symulation	Symulation	Główna klasa, która koordynuje całą symulacją. Tworzona jest struktura całego systemu. Zawiera pętle główna symulacji. Gromadzi statystyki.	Znajdują się w niej zmienne, które gromadzą informacje potrzebne do statystyk.
Source	Source	Klasa, która odpowiedzialna jest za planowanie nowych pakietów	
Packet	Packet	Klasa reprezentująca jedyny w systemie proces.	Zawiera zmienne potrzebne do przetwarzania statystyk.
Medium	Medium	Klasa reprezentująca medium transmisyjne. Zawiera informacje o tym czy medium transmisyjne jest wolne czy zajęte	medium_access, medium_time_busy
Transmitter	Transmitter	Klasa reprezentująca nadajnik. Zawiera w sobie bufor, w którym przechowywane są pakiety oczekujące na transmisje.	queue <Packet> bufor, negative_transmission_number_, positive_transmission_number_, packet_error_level_
Proces	Proces	Klasa z której dziedziczy klasa Packet i Source.	Funkcja wirtualna execute, Funkcja activate
Generator	Generator	Klasa reprezentująca generator liczb pseudolosowych.	Zawiera dwa rodzaje generatorów: - równomierny - wykładniczy
Event_List	Event_list	Klasa reprezentująca kalendarz zdarzeń.	Event_list agenda – gromadzi już zaplanowane zdarzenia, które są wykonywane po kolei w pętli głównej.
Link	Link	Klasa, której obiekty tworzą kalendarz zdarzeń,	Zawiera obiekty Klasy Event, które reprezentują zdarzenia.
Event	Event	Klasa reprezentująca zdarzenia.	Zdarzeniem jest zaplanowanie wygenerowanie nowego pakietu

4. Opis przydzielonej metody symulacyjnej

a. Schemat blokowy pętli głównej



Rys.2. Schemat blokowy pętli głównej.

b. Zdefiniowane procesy

W symulowanym systemie wyróżnia się jeden proces – Pakiet. W ramach procesu Pakiet wyróżnić można następujące fazy:

1. Pakiet pojawia się w systemie
   a. Zaplanowanie pojawienia się kolejnego pakietu
   b. Wstawienie pakietu do bufora

2. Sprawdzenie dostępu do łącza
   a. Jeśli jest dostęp do łącza przejdź do następnej fazy
   b. Jeśli brak dostępu do łącza uśpij proces na 0,5ms

3. Pakiet trafia do łącza (transmisja pakietu)
   a) Jeśli transmisja zakończyła się sukcesem prześlij potwierdzenie odbioru ACK i usuń pakiet z bufora
   b) Jeśli Nadajnik nie otrzymał ACK po czasie CTP + CTIZ retransmituj pakiet jeśli RL<15

4. Pakiet kończy obsługę
   a. Zwolnij łącze
   b. Jeśli jest pakiet oczekujący transmisji aktywuj ten proces
   c. Opuść system

5. Parametry wywołania programu

Każda symulacja zostaje wywołana z czasem symulacji równym 60000 ms oraz z ustawioną wartością lambda równą . Faza początkowa również jest ustawiona i jej wartość wynosi 1200 ms. Użytkownik sam wybiera numer symulacji którą chce zasymulować.

6. Generatory

a. Opis zastosowanych generatorów liczb losowych z histogramami

Generatory stworzone zostały na podstawie informacji pomocniczych. Poniżej przedstawiam kod generatorów:

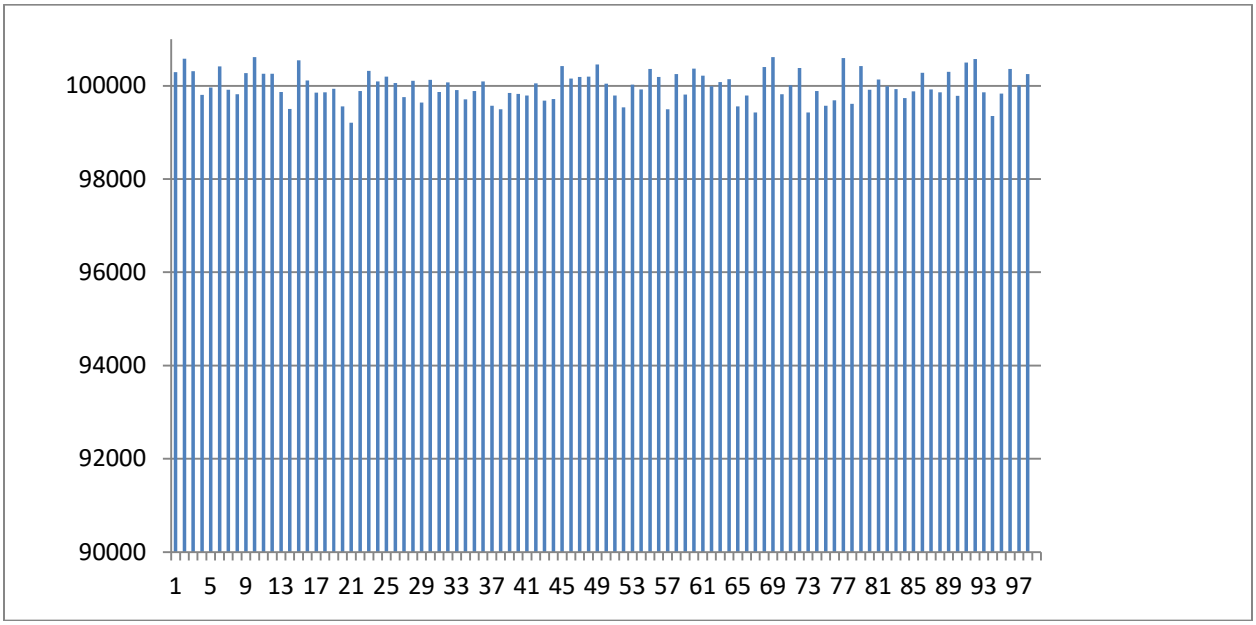
```
#ifndef TEST_LICZB_PSEUDOLOSOWYCH_GENERATOR_H
#define TEST_LICZB_PSEUDOLOSOWYCH_GENERATOR_H
class Generator
{
public:
	Generator(int kernel);
	~Generator();

	const double M = 2147483647.0;
	const double A = 16807;
	const double Q = 127773;
	const double R = 2836;
	int kernel_;
	Generator* generator_ptr;
	double generator_rownomierny();
	double generator_wykladniczy(double lambda);
};
#endif // SYMULACJA CYFROWA Generator H
```

```
#include "generator.h"
#include <cmath>
Generator::Generator(int kernel)
{
	kernel_ = kernel;
}
Generator::~Generator()
{
}
double Generator::generator_rownomierny()
{
	int h = kernel_ / Q;
	kernel_ = A * (kernel_ - Q*h) - R*h;
	if (kernel_ < 0) kernel_ = kernel_ + static_cast<int>(M);
	return static_cast<long double>(kernel_) / static_cast<long double>(M);
}
double Generator::generator_wykladniczy(double lambda)
{
	return -log(generator_rownomierny()) / lambda;
}
```

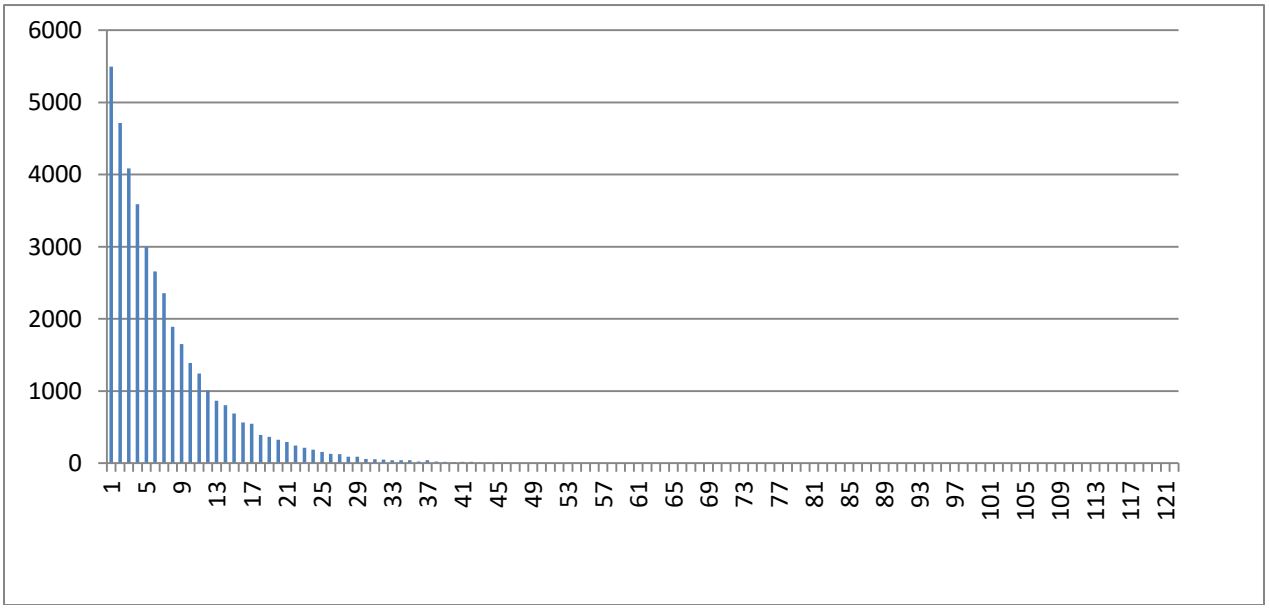
Testując generatory zapisałem wyniki w pliku tekstowym, następnie zaimportowałem wyniki do programu Microsoft Excel i wykonałem histogramy.

Poniżej prezentuje histogram generatora równomiernego:



Rys.3. Histogram generatora równomiernego.

Poniżej prezentuje histogram generatora wykładniczego:



Rys.4. Histogram generatora wykładniczego.

Poniżej prezentuje program, w którym testowałem generatory:

```
int main()
{
	cout << "Hello" << endl;
	fstream plik;
	Generator gen(123);
	int czas_generacji = 1000;
	/*while(czas_generacji--)
	{
		cout << gen.generator_rownomierny() << endl;
	}*/
	double lambda = 15;
	double tab[100]={0};
	czas_generacji = 10000000;
	while (czas_generacji--)
	{
		double x = gen.generator_wykladniczy(lambda)* 100;
		tab[static_cast<int>(x)]++;
		//cout << x << endl;
	}
	plik.open("plik_tekstowy_wykladniczy.txt", ios::out);
	for (int i = 1; i<99; i++) plik << tab[i] <<",";
	plik.close();
	system("pause");
	return 0;
}
```

b. Wyjaśnienie, w jaki sposób została zapewniona niezależność sekwencji losowych w różnych symulacjach

Do wygenerowania ziaren napisałem oddzielny krótki program, w którym wygenerowałem do pliku tekstowego 399 ziaren. Dla zapewnienia niskiej korelacji między ziarnami kolejne ziarna zapisywałem z odstępem 5 milionów próbek. Plik tekstowy nosi nazwę ziarna123.txt. Przewidziałem 10 symulacji. Ziaren wygenerowałem więcej, ponieważ podczas testów symulacji zmieniałem liczbę nadajników aby sprawdzić zachowanie się systemu. Ziarna z pliku tekstowego wczytuje do tablicy. Każda źródło potrzebuje 3 generatorów, z tego wynika że dla 10 nadajników w jednej symulacji potrzeba 30 generatorów. Dla wszystkich 10 symulacji potrzeba więc 300 generatorów.

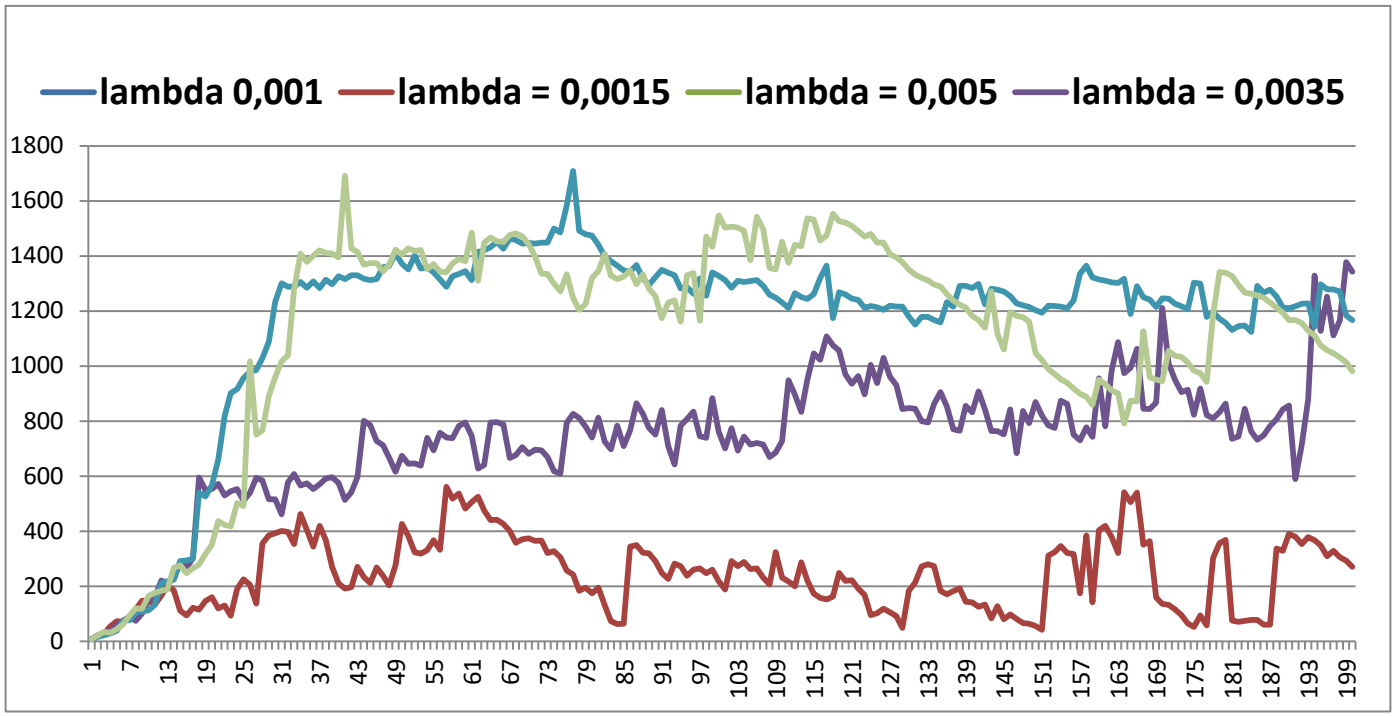
7. Krótki opis zastosowanej metody testowania i weryfikacji poprawności działania programu

Do weryfikacji poprawności działania programu użyłem pracy krokowej aby dokładnie sprawdzać co dzieje się po każdej pętli symulacyjnej w każdym buforze który jest w nadajniku. Miałem podgląd na kalendarz zdarzeń, liczbę wygenerowanych, retransmitowanych i utraconych pakietów po każdej pętli. Dokładniejszą metodą testowanie był tryb debugera. Ta metoda jest dużo trudniejsza i potrzeba dużo więcej czasu, ale można kontrolować zachowywanie się systemu linijka po linijce kodu. Ostatnim sposobem sprawdzania poprawności działania programu było zmienianie parametrów takich jak lambda, długość fazy początkowej, zmiana długości czasu symulacji i obserwacja wyników.

8. Wyniki symulacji

Dla systemu z czterema nadajnikami oraz możliwością retransmisji każdego pakietu aż 15 razy trudno było uzyskać zadawalające i przypominające rzeczywistość wyniki symulacji. Dla lepszych wyników liczba transmiterów została zwiększona do 10 oraz możliwa liczba retransmisji pojedynczego pakietu została zmniejszona do 5.

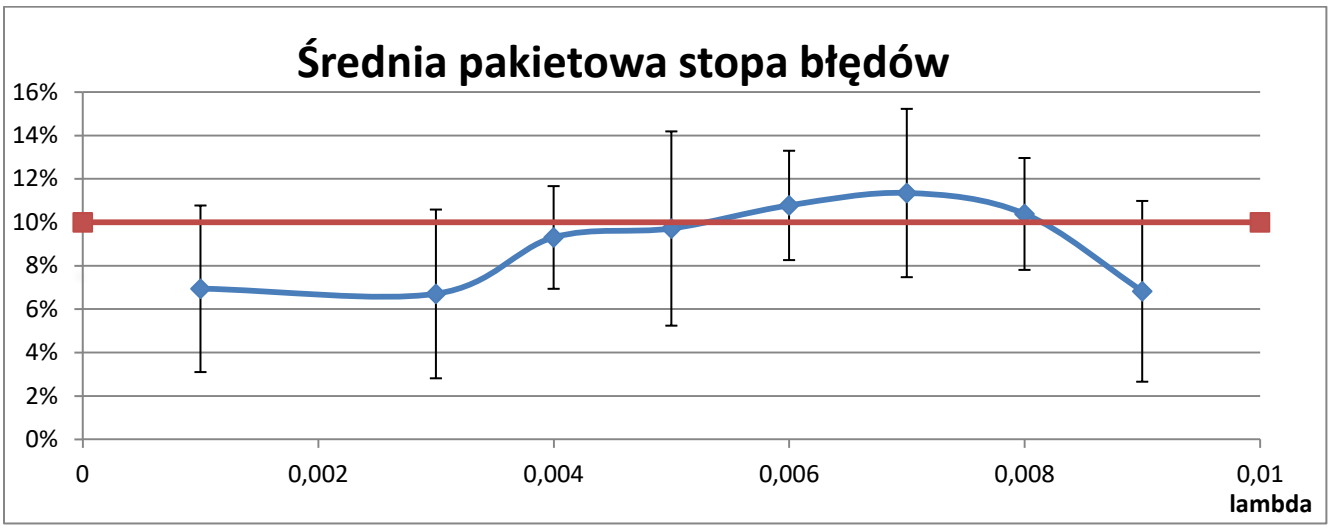
a. Wyznaczenie długości fazy początkowej



Rys.5. Wykres przedstawiający opóźnienie danego pakietu
dla różnych wartości lambda

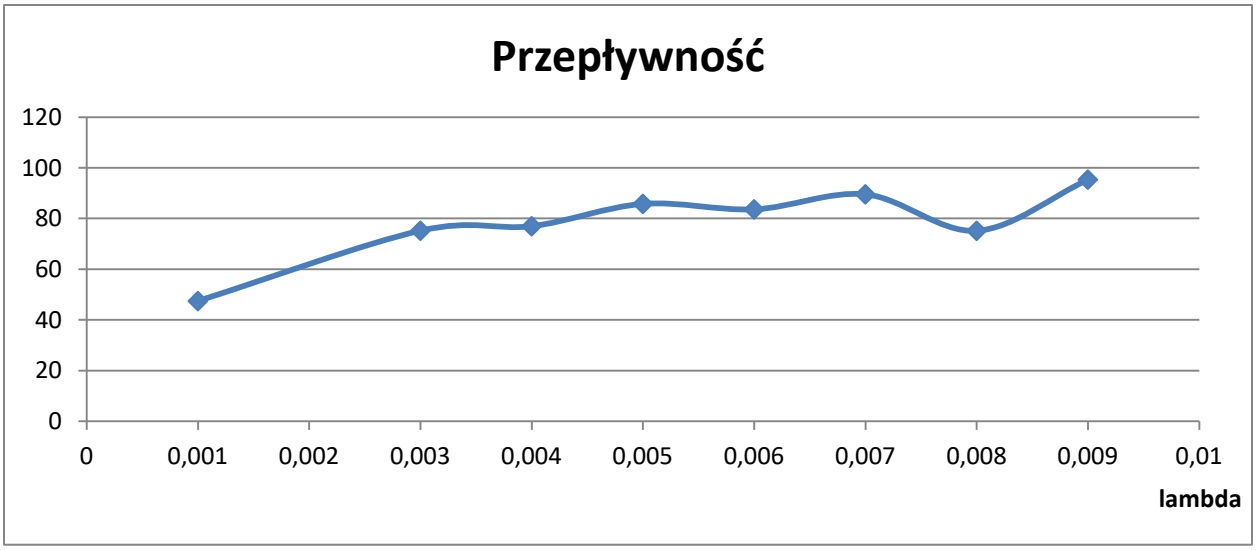
Na osi Y przedstawiony jest średni czas opóźnienia pakietu w ms po przeprowadzeniu 10 symulacji. Oś X przedstawia numer odebranego pakietu dla którego policzona została średnia. Jak łatwo zauważyć po około 30 dostarczonych pakietach system się stabilizuje. Fazę początkową ustawiam więc na wartość 1200 ms, ponieważ średnio dla 10 przeprowadzonych symulacji 30 dostarczony pakiet pojawia się po czasie 1200 ms.

b. Wyznaczenie wartości parametru lambda

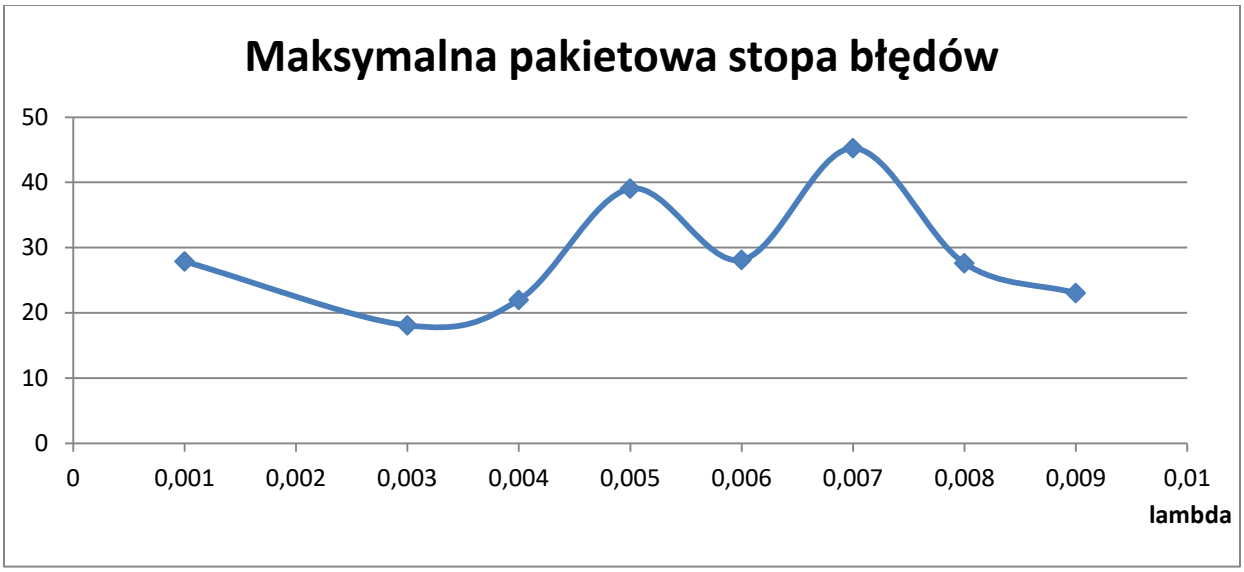


Rys.6. Wykres średniej pakietowej stopy błędów
dla różnej wartości lambda.

c. Wykres maksymalnej pakietowej stopy błędów i przepływności w zależności od wartości parametru lambda



Rys.6. Wykres przepływności dla różnej
wartości lambda



Rys.7. Wykres maksymalnej pakietowej stopy błędów
dla różnej wartości lambda

Jak widać z powyższych trzech wykresów najlepiej gdy wartość lambdy mieści się w okolicach 0,0035. Dla lambdy o takiej wartości średnia pakietowa stopa błędów nie przekracza 10%. Jest bardzo niska wartość maksymalnej pakietowej stopy błędów a przepływność jest na zadawalającym poziomie. Dodatkowo zmieniając delikatnie wartość lambdy parametry systemu nie zmienią się znacząco. Inaczej jest dla lambd o większych wartościach. Zbyt mała wartość lambdy co prawda powoduje małą stopę błędów, ale przepływność systemu jest niska.

d. Tabelka z wynikami symulacji dla każdego przebiegu symulacyjnego

Nr. symulacji	Średnia pakietowa stopa błędów	Maksymalna pakietowa stopa błędów	Średnia liczba retransmisji	Średni czas oczekiwania [ms]	Średnie opóźnienie [ms]	Przepływność [pakiet / sek]
1	8,159 %	15,436 %	0,183	17637,700	18038,9	74,800
2	7,131 %	21,622 %	0,099	15275,800	15443,8	87,283
3	11,047 %	22,619 %	0,449	21147,900	21750,8	75,167
4	9,558 %	19,540 %	0,184	13348,300	13779,7	71,017
5	12,588 %	21,324 %	0,161	14960,100	15156,7	94,866
6	10,598 %	20,000 %	0,385	15809,100	16420,6	66,983
7	12,978 %	50,000 %	0,144	16123,500	16450,6	83,033
8	9,591%	17,341%	0,580	20004,5	20774,0	75,450
9	9,374 %	16,783 %	0,298	17116,900	17501,0	75,100
10	2,027 %	16,000 %	0,034	14381,800	14527,7	66,033

e. Wyniki końcowe w postaci uśrednionych wyników po wszystkich przebiegach oraz przedziały ufności dla każdego z szczęściu parametrów

	Średnia pakietowa stopa błędów	Maksymalna pakietowa stopa błędów	Średnia liczba retransmisji	Średni czas oczekiwania [ms]	Średnie opóźnienie [ms]	Przepływność [pakiet / sek]
Średnia	**9,305 %**	**22,066 %**	**0,252**	**16579,780**	**16984,380**	**76,973**
Przedział ufności +/-	**2,365 %**	**7,673 %**	**0,130**	**1852,131**	**1962,601**	**6,797**

9. Wnioski

Podczas pisania dużo czasu zajęło mi ustalenie tego ile i jakich klas potrzebuje oraz zrozumieniu metody symulacyjnej. Dzięki notatkom z wykładu zdecydowanie lepiej zrozumiałem na czym polega metoda interakcji procesów. Wykorzystałem kilka rozwiązań, które zostały przedstawione na wykładzie. Początkowe założenia w moim projekcie przewidywały, że system będzie się składać z 4 nadajników oraz tego, że każdy pakiet może być retransmitowany 15 razy zanim może zostać uznany za odrzucony. Niestety dla tak małego systemu ciężko było uzyskać pożądane wyniki dlatego po wielu próbach różnych konfiguracji zdecydowałem się na zwiększenie liczby nadajników do 10 oraz zmniejszenia liczby retransmisji dla każdego pakietu do 5. Dla takich ustawień systemu przeprowadzane były wszystkie symulacje. Udało się uzyskać wartość parametru lambda tak aby zbliżyć Się do wartości średniej pakietowej stopy błędów na poziomie 10%. Podczas pracy z projektem dużo się nauczyłem dzięki temu, że projekt był dość złożony, zawierał sporo klas, które są ze sobą powiązane. Zdecydowanie lepiej potrafię teraz debugować program. Na początku musiałem sobie przypomnieć sporo rzeczy z programowania w C++, ponieważ ostatnio częściej miałem styczność z programowaniem w innych językach na przykład: Java lub C#.

10. Wydruk kodu programu jako załącznik do raportu

10.1. Funkcja główna

```
#include <iostream>
#include <cstdlib>
#include <string>
#include "symulation.h"
using namespace std;

int main()
{
    //srand(time(nullptr));
    int work_step_by_step = 0;

    bool show_messages = false;

    double required_simulation_time=60000;          // czas trwania symulacji
    double begining_phase_length = 1200;            //Wartosc wyznaczona na podstawie wykresow fazy poczatkowej...
    int symulation_number = 0;
    cout << "SYMULACJA CYFROWA" << endl << "Marcin Przepiorkowski " << endl << "Elektronika i Telekomunikacja" << endl << endl;;

    cout << "Jesli chcesz wybrac prace krokowa wcisnij 1" << endl << endl;
    cin >> work_step_by_step;
    if (work_step_by_step == 1) { show_messages = 1; }
    cout << "Wpisz nr. symulacji do wykonania: [ od 0 do 9 ] " << endl;
    cin >> symulation_number;
//  ********************************************************************************** **********
    Symulation symulation;

        //cout << "podaj wymagany czas symulacji: " << endl;
        //cin >> required_simulation_time;
        if (required_simulation_time < 0) required_simulation_time = 60000; // bezpiecznik
        symulation.start_symulation(required_simulation_time, work_step_by_step, show_messages, begining_phase_length, symulation_number);
        cout << endl << "-------------KONIEC SYMULACJI------ " << symulation_number << " ----" << endl;
        cout << endl << endl;
        cout << endl << endl;
        system("pause");

    return 0;
}
```

10.2. Klasa Symulation

```
#include "symulation.h"
#include "event_list.h"
#include "process.h"
#include "event.h"
#include <ctime>
#include "transmitter.h"
#include <iostream>
#include <windows.h>
#include "medium.h"
#include <ostream>
#include "source.h"
#include <string>
#include <fstream>
#include "generator.h"
using namespace std;

Symulation::Symulation()
{
    agenda = new Event_list();
    all_packets_ = 0;
    all_positive_transmission_ = 0;
    all_negative_transmission_ = 0;
    all_retransmissions_ = 0;
    symul_ptr = this;
    all_delay_packet_time_ = 0.0;
    all_waiting_packet_time = 0.0;
    average_delay_packet_time_ = 0.0;
    average_waiting_packet_time = 0.0;
    all_positive_delivery_retransmition_ = 0.0;
    average_packet_error_level = 0.0;
    Generator ** generators = nullptr;
}

Symulation::~Symulation()
{
}

void Symulation::start_symulation(double requied_simulation_time, int
  work_step_by_step,  bool show_messages, double begin_phase_length, int
  sym_numb)
{

    generators = new Generator*[400];
    fstream file;
    file.open("ziarna123.txt", ios::in);
    if (file.good() == false)
    {
        cout << "Plik nie istnieje";
        exit(0);

    }
    string line;
    int licznik = 0;
    while (getline(file, line))
    {
        generators[licznik] = new Generator(atoi(line.c_str()));
        licznik++;
    }
    file.close();
```

```
57
58
59      HANDLE hOut;
60      hOut = GetStdHandle(STD_OUTPUT_HANDLE);
61  //---------------------------------------------------------------------------
62      double clock = 0.0;
63      Medium *medium=new Medium();
64      Transmitter transmitter[10];            // stworzenie 10 transmiterow
          (nadajnikow)
65      Transmitter *tran_tab_ptr = transmitter;
66                                              // ZMIANA LICZBY
                    TRANSMITEROW.......*************************************
                    **************************** z 4 na 10
67      for (int i = 0; i < 10;++i)
68      {
69          transmitter[i].transmiter_id_=i;
70          transmitter[i].packet_generation_number = 0;
71      }
72
73      Transmitter *transmitter_ptr = nullptr;
74      double max_packet_error_level;
75
76      transmitter_ptr = &transmitter[0];
77  Source *source0 = new Source(transmitter_ptr,symul_ptr,sym_numb*30+0);
78  source0->start_source(agenda, clock);
79
80      transmitter_ptr = &transmitter[1];
81  Source *source1 = new Source(transmitter_ptr, symul_ptr, sym_numb * 30 + 3);
82  source1->start_source(agenda, clock);
83
84      transmitter_ptr = &transmitter[2];
85  Source *source2 = new Source(transmitter_ptr, symul_ptr, sym_numb * 30 + 6);
86  source2->start_source(agenda, clock);
87
88      transmitter_ptr = &transmitter[3];
89  Source *source3 = new Source(transmitter_ptr, symul_ptr, sym_numb * 30 + 9);
90  source3->start_source(agenda, clock);
91  //
      ***************************************************************************
      *******************************************
```

```
92  transmitter_ptr = &transmitter[4];
93  Source *source4 = new Source(transmitter_ptr, symul_ptr, sym_numb * 30 + 12);
94  source4->start_source(agenda, clock);
95
96  transmitter_ptr = &transmitter[5];
97  Source *source5 = new Source(transmitter_ptr, symul_ptr, sym_numb * 30 + 15);
98  source5->start_source(agenda, clock);
99
100 transmitter_ptr = &transmitter[6];
101 Source *source6 = new Source(transmitter_ptr, symul_ptr, sym_numb * 30 + 18);
102 source6->start_source(agenda, clock);
103
104 transmitter_ptr = &transmitter[7];
105 Source *source7 = new Source(transmitter_ptr, symul_ptr, sym_numb * 30 + 21);
106 source7->start_source(agenda, clock);
107 //
      ***************************************************************************
      *******************************************
108 transmitter_ptr = &transmitter[8];
109 Source *source8 = new Source(transmitter_ptr, symul_ptr, sym_numb * 30 + 24);
110 source8->start_source(agenda, clock);
111
112 transmitter_ptr = &transmitter[9];
113 Source *source9 = new Source(transmitter_ptr, symul_ptr, sym_numb * 30 + 27);
114 source9->start_source(agenda, clock);
```

```
119
120
121
122     while(clock < requied_simulation_time)
123     {
124
125         //-------------------------------------------------------//
126         SetConsoleTextAttribute(hOut, FOREGROUND_RED);
127 if(show_messages) cout << endl << "\t AGENDA" << endl;
128 if (show_messages)agenda->show_agenda();                          //to dla mnie
      info o agendzie
129         current = agenda->first()->proc_;
130         clock = agenda->first()->event_time_;
131
132         agenda->remove_first(show_messages);
133         current->execute(agenda, clock,medium, tran_tab_ptr,
              symul_ptr,begin_phase_length);      // wykonywanie pierwszego pakietu
                z agendy


134
135         SetConsoleTextAttribute(hOut, FOREGROUND_INTENSITY);
136 if (show_messages)  cout << "Wykonywanie procesu: " << current <<"\t faza
      procesu: "<<current->phase<< "\t id_pakietu (procesu): "<< "POMYSLE
      POZNIEJ"<<endl;
137
138         SetConsoleTextAttribute(hOut, FOREGROUND_GREEN);
139 if (show_messages)  cout <<"clock: "<< clock <<"\t requied_simulation_time:
      "<<requied_simulation_time<<endl<<endl;
140
141         for (int i = 0; i < 10; i++)
142         {
143 if (show_messages)  cout << "Bufor nr: " << i << "  ilosc pakietow w buforze: "
      << transmitter[i].bufor_.size() << endl;
144         }
145 if (show_messages)  cout << endl <<endl << endl;
146         if(work_step_by_step == 1)
147         {
148             system("pause");
149         }
150


156         if(begin_phase_length>clock)
157         {
158             set_all_packets(0);
159             set_all_retransmission(0);
160             set_all_positive_transmission(0);
161             set_all_negative_transmission(0);
162             set_all_delay_packet_time(0);
163             set_all_waiting_packet_time(0);
164             set_average_waiting_packet_time(0, 1);
165             set_averag_delay_time(0,1);
166             average_packet_error_level = 0.0;
167             all_positive_delivery_retransmition_ = 0.0;
168         }
```

```
169
170             if (show_messages)
171             {
172             cout << endl << "Liczba wygenerowanych pakietow: " << symul_ptr-
                  >all_packets_ << endl;
173             cout << "Liczba pozytywnych transmisji:  " << symul_ptr-
                  >all_positive_transmission_ << endl;
174             cout << "Liczba negatywnych transmisji:  " << symul_ptr-
                  >all_negative_transmission_ << endl;
```

```
175             cout << "Liczba retransmisji:  " << symul_ptr->all_retransmissions_
                   << endl << endl;
176             }
177
178     }//---------------------------------------------- END WHILE
179
180
181
182
183
184
185
186
187 cout << endl << endl <<" S T A T Y S T Y K I  DLA SYMULACJI nr. " <<sym_numb<<
      endl << endl;
188     for (int i = 0; i < 10; i++)
189     {
190         cout << "Bufor nr: " << i << "  ilosc pakietow w buforze: " <<
              transmitter[i].bufor_.size() <<endl<< endl;
191     }
192
193
194     cout << "Liczba wygenerowanych pakietow: " << symul_ptr->all_packets_ <<
        endl;
195     cout << "Liczba pozytywnych transmisji:  " << symul_ptr-
        >all_positive_transmission_ << endl;
196     cout << "Liczba negatywnych transmisji:  " << symul_ptr-
        >all_negative_transmission_ << endl;
197     cout << "Liczba retransmisji:  " << symul_ptr->all_retransmissions_ << endl
        << endl;
198
199
200         for (int i = 0; i < 10;i++)  //to dla mnie zeby wiedziec czy statystyki
              nei klamia :D
201         {
202             if(transmitter[i].show_positive_transmission_number() !=0)    //
                  zabezpieczenie --> czasem transmiter nie wyslal jeszcze nic i
                  byloby dzielenie przez 0 !!!
203             {
204                 transmitter[i].set_packet_error_level(transmitter
                      [i].show_negative_transmission_number(), transmitter
                      [i].show_positive_transmission_number());
205             }
206             else { transmitter[i].set_packet_error_level(0, 1); }
207             SetConsoleTextAttribute(hOut, FOREGROUND_RED);
208             cout << "\t Pakietowa Stopa bledow transmitera " << transmitter
                  [i].transmiter_id_ << "\t" <<transmitter
                  [i].show_packet_error_level() <<" %"<<
                  endl;SetConsoleTextAttribute(hOut, FOREGROUND_GREEN);
```

```
209             cout << "\t Liczba wygenerowanych pakietow przez transmiter: " <<
                  transmitter[i].packet_generation_number << endl;
210             cout << "\t Liczba dostarczonych pakietow przez transmiter: " <<
                  transmitter[i].show_positive_transmission_number() << endl<<endl;
211         }cout << endl << endl;
212         for (int i = 0; i < 10;i++)
213         {
214             average_packet_error_level += transmitter
                  [i].show_packet_error_level();
215         }
216         average_packet_error_level = average_packet_error_level / 10.0;
217         cout << "Srednia pakietowa stopa bledow: " << symul_ptr-
              >average_packet_error_level <<" %"<< endl;
218
219
220         max_packet_error_level = transmitter[0].show_packet_error_level();
221         for (int i = 1; i < 10; i++)
222         {
223             if (transmitter[i].show_packet_error_level() >
                  max_packet_error_level)
224             {
225                 max_packet_error_level = transmitter[i].show_packet_error_level
                      ();
226             }
227         }
228
229         cout <<endl<<endl<< "Maksymalna pakietowa stopa bledow: "
              <<max_packet_error_level<<" %"<<endl<< endl;
230
231         cout << "Srednia liczba retransmisji pakietow: " << symul_ptr-
              >all_positive_delivery_retransmition_ / symul_ptr-
              >show_all_positive_transmissions() << endl << endl;
232
233         cout << "Przeplywnosc systemu: " << symul_ptr-
              >all_positive_transmission_ / requied_simulation_time *1000
              <<" [pakiet/sek]" <<endl<< endl;
234
235         symul_ptr->set_averag_delay_time(symul_ptr->show_all_delay_packet_time
              (), symul_ptr->show_all_positive_transmissions());
236         cout << "Srednie opoznienie pakietu w buforze: " <<  symul_ptr-
              >show_average_delay_time()<< " ms" << "\t\t";
237               cout << " all delay packet time: " << symul_ptr-
                    >show_all_delay_packet_time() << "\t\t all positive
                    transmission: " << symul_ptr->show_all_positive_transmissions
                    () <<endl<< endl;
238
239         symul_ptr->set_average_waiting_packet_time(symul_ptr-
              >show_all_waiting_time(), symul_ptr->show_all_positive_transmissions
              ());
240         cout << "Sredni czas oczekiwania: " << symul_ptr-
              >show_average_waiting_packet_time() << " ms" << "\t\t";
241               cout << " waiting time: " << symul_ptr->show_all_waiting_time()
                    << "\t\t all positive transmission: " << symul_ptr-
                    >show_all_positive_transmissions() <<endl<< endl;
242
243     cout << endl;
244     cout << endl << "ZEGAR --> " << clock << endl;
245
246
247
248
249 }
250
```

```
251 void Symulation::set_all_packets(int x)
252 {
253     all_packets_ = x;
254 }
255
256 void Symulation::set_all_retransmission(int x)
257 {
258     all_retransmissions_ = x;
259 }
260
261 void Symulation::set_all_positive_transmission(int x)
262 {
263     all_positive_transmission_ = x;
264 }
265
266 void Symulation::set_all_negative_transmission(int x)
267 {
268     all_negative_transmission_ = x;
269 }
270
271
272
273 void Symulation::all_packets_increment()
274 {
275     ++all_packets_;
276 }
277
278 void Symulation::all_retrensmissions_increment()
279 {
280     ++all_retransmissions_;
281 }
282
283 void Symulation::all_positive_transmission_increment()
284 {
285     ++all_positive_transmission_;
286 }
287
288 void Symulation::all_negative_transmission_increment()
289 {
290     ++all_negative_transmission_;
291 }
292
293 void Symulation::set_all_delay_packet_time(double time)
294 {
295
296     all_delay_packet_time_ += time;
297 }
298
299 double Symulation::show_all_delay_packet_time()
300 {
301     return all_delay_packet_time_;
302 }
303
304 int Symulation::show_all_positive_transmissions()
305 {
306     return all_positive_transmission_;
307 }
308
309 void Symulation::set_averag_delay_time(double time1, int deliver_packets)
310 {
311      average_delay_packet_time_ =  time1 / static_cast<double>
       (deliver_packets);
312 }
313
```

```
314 double Symulation::show_average_delay_time()
315 {
316     return average_delay_packet_time_;
317 }
318 void Symulation::set_average_waiting_packet_time(double time1, int
      deliver_packets)
319 {
320     average_waiting_packet_time = time1 / static_cast<double>(deliver_packets);
321 }
322
323 double Symulation::show_average_waiting_packet_time()
324 {
325     return average_waiting_packet_time;
326 }
327
328 void Symulation::set_all_waiting_packet_time(double time)
329 {
330
331     all_waiting_packet_time += time;
332 }

333
334 double Symulation::show_all_waiting_time()
335 {
336     return all_waiting_packet_time;
337 }
338
```

Plik .h

```
1  #ifndef SYMULACJA_CYFROWA_SYMULATION_H
2  #define SYMULACJA_CYFROWA_SYMULATION_H
3  #include  "event_list.h"
4  #include <queue>
5  using namespace  std;
6  class Event_list;
7  class Process;
8  class Generator;
9
10 class Symulation
11 {
12     int all_packets_;
13     int all_retransmissions_;
14     int all_positive_transmission_;
15     double all_negative_transmission_;
16     double all_delay_packet_time_;
17     double all_waiting_packet_time;
18     double average_delay_packet_time_;
19     double average_waiting_packet_time;
20 public:
21     Symulation();
22     ~Symulation();
23     double all_positive_delivery_retransmition_;
24     Generator ** generators;
25     Process *current;
26     Event_list *agenda;
27     Symulation *symul_ptr;
28     double clock;
29     queue<double> my_packets_queue;
30     double average_packet_error_level;
31
32     void start_symulation(double required_simulation_time_,int
         work_step_by_step, bool show_messages,double begin_phase_length, int
         sym_numb);
```

```
33      void all_packets_increment();
34      void all_retransmissions_increment();
35      void all_positive_transmission_increment();
36      void all_negative_transmission_increment();
37      void set_all_delay_packet_time(double time);
38      void set_all_waiting_packet_time(double time);
39      double show_all_delay_packet_time();
40      int show_all_positive_transmissions();
41      double show_all_waiting_time();
42      void set_averag_delay_time(double time1, int deliver_packets);
43      void set_average_waiting_packet_time(double time1, int deliver_packets);
44      double show_average_delay_time();
45      double show_average_waiting_packet_time();
46      void set_all_packets(int x);
47      void set_all_retransmission(int x);
```

```
48      void set_all_positive_transmission(int x);
49      void set_all_negative_transmission(int x);
50  };
51  #endif // SYMULACJA CYFROWA Symulation H
52
53
```

10.3. Klasa Source

```
1  #include "source.h"
2  #include "packet.h"
3  #include <ctime>
4  #include "transmitter.h"
5  #include "symulation.h"
6  #include "generator.h"
7  #include <iostream>
8  Source::Source()
9  {
10 }
11 Source::Source(Transmitter* ptr, Symulation* sym_ptr, int index)
12 {
13     transmiter_ptr = ptr;
14     symulation_ptr = sym_ptr;
15     index_ = index;
16 }
17 Source::~Source()
18 {
19 }
20 void Source::execute(Event_list *list, double clock, Medium *med, Transmitter
     *ptr, Symulation *sym_ptr, double begin_phase_lengt)
21 {
22     int time1 = symulation_ptr->generators[index_+1]->generator_rownomierny()
       *10 +1;  // CTP [1,2,...10]
23     int temp = symulation_ptr->generators[index_]->generator_wykladniczy(0.004);
           // CGP rozklad wykladniczy
24     double time2 = temp / 10.0;
       // CGP rozklad wykladniczy cd.

25     Packet *temp_ptr;
26     (temp_ptr = new Packet(transmiter_ptr, symulation_ptr,this,time1));  //
       wygenerowanie nowego pakietu
27     temp_ptr->activate(0.0, list, clock);           // umieszczenie na
       agendziehtd
28     transmiter_ptr->generate_packet_(temp_ptr, clock);  // wrzucenie do bufora
29     activate(time2,list, clock);        // zaplanowanie stworzenia nowego
       pakietu
30 }
```

```
31 void Source::start_source(Event_list *list, double clk)
32 {
33     double time = symulation_ptr->generators[index_]->generator_wykladniczy
         (0.75);  // tu moze byc co chce bo to i tak tylko raz sie wykonuje na
         starcie -- faza poczatkowa
34     int temp_time = time * 10;
35     time = temp_time / 10.0;
36     activate(time, list, clk);  // umieszczenie na agendzie
37     cout <<"czas w start source  "<<this->transmiter_ptr->transmiter_id_<<"\t"<<
         time << endl;
38 }
```

Plik .h

```
1  #ifndef SYMULACJA_CYFROWA_SOURCE_H
2  #define SYMULACJA_CYFROWA_SOURCE_H
3  #include "process.h"
4
5  class Generator;
6
7  class Source:public Process
8  {
9  public:
10     Source();
11     ~Source();
12     Source(Transmitter *ptr, Symulation *sym_ptr,int index);
13
14
15     Transmitter  *transmiter_ptr;
16     Symulation   *symulation_ptr;
17     Generator    *generator_ptr;
18     int index_;  //  numer indeksu w tablicy do ziarna
19     void execute(Event_list *list, double clock, Medium *med, Transmitter *ptr,
         Symulation *sym_ptr, double begin_phase_lengt);
20     void start_source(Event_list *list, double clk);
21
22
23 };
24 #endif // SYMULACJA CYFROWA Source H
25
```

10.4.Klasa Medium

```
1  #include "medium.h"
2  Medium::Medium()
3  {
4      medium_access = true;
5      collision = false;
6      medium_time_busy =0;
7      iterator = 0;
8      for (int i = 0; i < 10;i++)
9      {
10         tab_transmitted_packets[i] = nullptr;
11     }
12 }
13 Medium::~Medium()
14 {
15 }
16
```

```cpp
bool Medium::medium_free()
{
    return medium_access;
}
bool Medium::set_medium_access(bool data)
{

    return medium_access = data;
}
double Medium::show_medium_time_busy()
{
    return medium_time_busy;
}
double Medium::set_medium_time_busy(double busy_time,double clk)
{
    return medium_time_busy = busy_time+clk;
}
bool Medium::show_medium_access()
{
    return medium_access;
}
```

Plik .h

```cpp
#ifndef SYMULACJA_CYFROWA_MEDIUM_H
#define SYMULACJA_CYFROWA_MEDIUM_H
class Packet;

class Medium
{
    bool medium_access; // dostep do lacza
    double medium_time_busy; // czas zajetosci medium;


public:
    Medium();
    ~Medium();

    Packet *tab_transmitted_packets[10];  // tablica przesylanych paietow
                                          // jesli jest wiecej niz jeden element
                    -->  KOLIZJA MOZLIWA
    int iterator;                        //pomocny do w/w tablicy
    bool collision;

    bool medium_free();
    bool set_medium_access(bool data);
    double set_medium_time_busy(double busy_time,double clk);
    double show_medium_time_busy();

    bool show_medium_access();
    //double difs;
};
#endif // SYMULACJA CYFROWA Medium H
```

10.5. Klasa Packet

```
#include "packet.h"
#include <iostream>
#include <windows.h>
#include "medium.h"
#include "process.h"
#include "transmitter.h"
#include "source.h"
#include "generator.h"
using namespace std;

Packet::Packet(Transmitter *ptr, Symulation *sym_ptr,Source *source_ptr, double time)
{
    retransmission_number_ = 0;   // LR
    ack_transmission_time_ = 1;   // CTIZ
    packet_transmission_time_ = time;
    packet_id = 0;
    double current_difs = 0.0;
    transmiter_ptr = ptr;
    symulation_ptr = sym_ptr;
    src_ptr_ = source_ptr;
    sym_ptr->all_packets_increment();
    delivery_time_packet_ = 0.0;
    delay_packet_timie_ = 0.0;
    arrival_packet_time = sym_ptr->clock;
    leave_bufor_packet_time = 0.0;
    waiting_packet_time = 0.0;
    time_in_medium = 0.0;
}


Packet::~Packet()
{
}
void Packet::set_ptt(int cgp)
{
    packet_transmission_time_ = cgp;
}

void Packet::inc_ret_number()
{
    ++retransmission_number_;
}
void Packet::set_packet_id(int transmit, int numb)
{
    packet_id = numb*10+transmit;  // zakodowany id pakietu = nr. wygenerowanego pakietu prze i'ty transmiter*10 + id transmitera
}
int Packet::show_packet_id()
{
    return packet_id;
}

void Packet::set_arrival_packet_time(double time)
{

    arrival_packet_time = time;
}
double Packet::show_arrival_packet_time()
{
    return arrival_packet_time;
}

```

```
62 void Packet::set_leave_bufor_packet_time(double time)
63 {
64
65     leave_bufor_packet_time = time;
66 }
67 double Packet::show_leave_bufor_packet_time()
68 {
69     return leave_bufor_packet_time;
70 }
71
72 void Packet::set_delay_packet_time(double time)
73 {
74
75     delay_packet_timie_ = time;
76 }
77
78 void Packet::set_waiting_packet_time(double time)
79 {
80
81     waiting_packet_time = time;
82 }
83 void Packet::set_delivery_packet_time(double time)
84 {
85
86     delivery_time_packet_ = time;
87 }
88
89
90
91
92 void Packet::execute(Event_list *list, double clock, Medium *med, Transmitter
     *ptr, Symulation *sym_ptr, double begin_phase_lengt)
93 {
94
95  bool show_messages = false;  //**************************          to zmienna
    ------------------------------------------------------------------------------
       do doatkowa potrzeba do debugowania
96
97      if (show_messages)
98      {
99          system("cls");
100         cout << "Symulacja w trakcjie... Porosze czekac :) " << endl;
101         cout << "clock: " << clock << "\t\t liczba retransmisji aktualnie
              wykonujacego sie pakietu: " << this->retransmission_number_;
102     }
103
104
105     HANDLE hOut;
106     hOut = GetStdHandle(STD_OUTPUT_HANDLE);
107     SetConsoleTextAttribute(hOut, FOREGROUND_GREEN);
108     //
          ********************************************************************************
          ****************************************
109     if (show_messages) {
110         cout << "Witaj w funkcji execute klasy Packet, zaczynmy zabawe :D " <<
              endl;
111         cout << "czas zajetosci medium: " << med->show_medium_time_busy() <<
              "\t clock: " << clock << endl;
112         cout << "dostep do lacza: " << med->show_medium_access() << endl;
113         cout << "Pakiet wygenerowany z Transmitera nr. " << this-
              >transmiter_ptr->transmiter_id_ << "\t ID pakietu:" << this-
              >packet_id << "\t Liczba retransmisji pakietu: " << this-
              >retransmission_number_ << endl;
114     }
```

```
115     bool active = true;
116     while (active)
117     {
118         switch (phase)
119         {
120         case 0:
121             if (show_messages)  cout << "switch 0" << endl;
122             current_difs = 0.0;
123             if (!this->transmiter_ptr->bufor_.empty())
124             {
125                 if (transmiter_ptr->bufor_.front()->packet_id == this->packet_id)     //sprawdzenie czy pakiet jest pierwszy w kolejce do wysylania w buforze
126                 {
127                     this->set_leave_bufor_packet_time(clock); // ustawienie czasu opuszczenia pakietu z bufora
128                     phase = 1;            // jesli tak to idz do kolejej fazy
129                 }
130                 else
131                 {
132                     active = false; break;      //uspij sie
133                 }
134             }
135         case 1:         // nasluchiwanie kanalu --> if DIFS >2ms  && medium jest wolne  -->faza 2
136             if (show_messages)  cout << "switch 1" << endl;
137             if (clock >= med->show_medium_time_busy())      //czy medium powinno byc juz wolne?
138             {
139                 med->set_medium_access(true);
140                 if (show_messages)  cout << "NASLUCHIWANIE WOLNEGO LACZA...." << "DIFS = " << current_difs << endl;
141                 if (current_difs > 2)
142                 {
143                     this->time_in_medium = clock;
144                     if (show_messages)      cout << "Mozna nadawac.. !!! --> DIFS = " << current_difs << endl;
145                     med->tab_transmitted_packets[med->iterator] = this;
146                     med->iterator = (med->iterator + 1) % 10;       // bylo mod 4
147                     if (show_messages)      cout << "adres pakietu: " << this << endl;   //*********
148                     phase = 2;
149                     this->activate(0.0, list, clock); active = false;
150                     break;
151                 }
152                 current_difs += 0.5;
153                 this->activate(0.5, list, clock);
154                 active = false; break;
155             }
156             this->activate(0.5, list, clock);
157             current_difs = 0.0;
158             active = false;
159             break;
160         case 2:
161             if (med->iterator >1) //jesli iterator >2 --> sa co najmniej 2 pakiety w medium.. jest wiecej niz 1 pakiet --> trzeba sprawdzic czy czasy sa takie same czy nie
162             {
163                 for (int i = 0; i < 10; i++)
164                 {
165                     if (med->tab_transmitted_packets[i % 20] != nullptr && med->tab_transmitted_packets[(i + 1) % 10] != nullptr) //zabezpieczenie przed przekroczeniem tablicy
166                     {
167                         if (med->tab_transmitted_packets[i % 10]->time_in_medium == med->tab_transmitted_packets[(i + 1) % 10]->time_in_medium) // czy czasy te same
168                         {
169                             med->collision = true;
```

```
170                         }
171                     }
172                 }
173                 if (med->collision)     // KOLIZJA WYKRYTA --> Retransmisja !!
174                 {
175                     for (int i = 0; i < 10; i++)
176                     {
177                         if (med->tab_transmitted_packets[i % 10] != nullptr)
178                         {
179                             if (med->tab_transmitted_packets[i % 10]-
                        >retransmission_number_>=5)
180                             {
181                                 med->tab_transmitted_packets[i % 10]->phase =
                        4;
182                                 active = false; break;
183                             }
184                             med->iterator = 0; //wyzerowanie iteratora
185                             med->tab_transmitted_packets[i]->phase = 1;
186                             med->tab_transmitted_packets[i]-
                        >retransmission_number_++;
187                             sym_ptr->all_retransmissions_increment();
188                             int max_range = static_cast<int>(pow(2.0, this-
                        >retransmission_number_)-1);            // losowanie czasu
                        ponownej retransisji
189                             double R = symulation_ptr->generators[src_ptr_-
                        >index_ + 2]->generator_rownomierny() * 100;
190                             int temp_R = static_cast<int> (R);
191                             R = temp_R *max_range;
192                             R = R / 10.0;
193                             med->tab_transmitted_packets[i]->activate(med-
                        >tab_transmitted_packets[i]->packet_transmission_time_*R,
                        list, clock); //RETRANSMISJA
194                             med->tab_transmitted_packets[i] = nullptr;
195                         }
196                     }
197                     active = false;
198                     break;
199                 }
200             }
201             else // tylko jeden pakiet w medium --> wysylaj
202             {
203                 med->iterator = 0; //wyzerowanie iteratora
204                 this->activate(this->packet_transmission_time_, list, clock);
205                 med->set_medium_access(false);
206                 med->set_medium_time_busy(this->packet_transmission_time_ +
                    this->ack_transmission_time_, clock);
207                 this->phase = 3;
208                 active = false;
209                 break;
210             }
211         case 3:     //  Koniec transmisji
212                                                     if (show_messages)
                            cout << "switch 3" << endl;
213                                                     if (show_messages)
                            cout << "Koniec transmisji pakietu ..." << endl;
214             if (retransmission_number_ < 5)              // ZMIANA LICZBY
                    MOZLIWYCH RETRANSMISJI DO 5 Z 15 !!!!!!!!!!!!!!!!!!!
215             {
216                     phase = 4;
217                     transmiter_ptr->ack = true;     // ustaienie potwierdzenia
                        transmisji ACK
218             }
219             else
220             {
221                 transmiter_ptr->ack = false;
222                 terminated = true;
223                 phase = 4;
224             }
```

```
225         case 4:     // koniec transmisji...
226             if (show_messages)  cout << "switch 4" << endl;
227             if (show_messages)  cout << "KONIEC TRANSMISJI !!!!" << endl;
228             med->tab_transmitted_packets[0] = nullptr; med->iterator = 0;
229                             for (int i = 0; i < 10; i++)
230                             {
231                                 if (show_messages)      cout << med-
                    >tab_transmitted_packets[i] << endl;
232                             }
233             if (transmiter_ptr->ack)  // jesli odebrano ACK --> sukces
234             {
235                 SetConsoleTextAttribute(hOut, FOREGROUND_RED);
236                 if (show_messages)  cout << "TRANSMISJA ZAKONCZONA
                  SUKCESEM !!!" << endl;
237                 this->transmiter_ptr->positive_transmission_number_increment();
238
239                 sym_ptr->all_positive_transmission_increment();
240                 if (this->retransmission_number_ > 1) sym_ptr-
                  >all_positive_delivery_retransmition_++;
241
242                 this->set_delivery_packet_time(clock);
243                 if(begin_phase_lengt<clock) this->set_delay_packet_time
                  (delivery_time_packet_ - arrival_packet_time);
244                 this->set_waiting_packet_time(leave_bufor_packet_time -
                  arrival_packet_time);
245                 sym_ptr->set_all_delay_packet_time(delay_packet_timie_);
246                 sym_ptr->set_all_waiting_packet_time(waiting_packet_time);
247                 if (!this->transmiter_ptr->bufor_.empty())
248                 {
```

```
249                     this->transmiter_ptr->bufor_.pop();     //  zdejmowanie
                    dostarczonego pakietu z bufora
250                     if (!this->transmiter_ptr->bufor_.empty())
251                     {
252                         this->transmiter_ptr->bufor_.front()->activate(0.0,
                    list, clock);       // obudenie kolejnego pakietu w buforze
253                     }
254                 }
255                 SetConsoleTextAttribute(hOut, FOREGROUND_GREEN);
256                 active = false; terminated = true;
257                 break;
258             }
259             SetConsoleTextAttribute(hOut, FOREGROUND_BLUE);
260             if (show_messages)  cout << "TRANSMISJA ZAKONCZONA
                NIEPOWODZENIEM ...." << endl;
261             this->transmiter_ptr->negative_transmission_number_increment();
                    // inkrementacja odrzuconych pakietow przez transmiter
262             sym_ptr->all_negative_transmission_increment();
263             if (show_messages)  cout << this->retransmission_number_ << endl;
264             SetConsoleTextAttribute(hOut, FOREGROUND_GREEN);
265             if (!this->transmiter_ptr->bufor_.empty())
266             {
267                 this->transmiter_ptr->bufor_.pop(); //  zdejmowanie utraconego
                  pakietu z bufora
268                 if (!this->transmiter_ptr->bufor_.empty())
269                 {
270                     this->transmiter_ptr->bufor_.front()->activate(0.0, list,
                    clock);   // obudenie kolejnego pakietu w buforze
271                 }
272             }
273             terminated = true;
274             active = false;
275             break;
276             } //end switch
277
278         }
```

```
279             if(begin_phase_lengt>clock)
280             {
281                 set_arrival_packet_time(0);
282                 set_leave_bufor_packet_time(0);
283                 set_delivery_packet_time(0);
284                 set_delay_packet_time(0);
285                 set_waiting_packet_time(0);
286                 this->transmiter_ptr->set_negative_transmission_number(0);
287                 this->transmiter_ptr->set_positive_transmission_number(0);
288                 this->symulation_ptr->all_positive_delivery_retransmition_ =
                      0;
289             }
290     }
```

Plik .h

```
1  #ifndef SYMULACJA_CYFROWA_PACKET_H
2  #define SYMULACJA_CYFROWA_PACKET_H
3  #include "process.h"
4  #include "transmitter.h"
5  class Medium;
6  class Source;
7
8  class Packet:public Process  // klasa Pakiet jest jedynym procesem w tej symulce
9  {
10     double packet_transmission_time_;   // czas transmisji pakietu
11     int retransmission_number_;     // liczba retransmisji
12     int ack_transmission_time_; // czas transmisji ACK  CTIZ =1ms
13     int packet_id;
14     double arrival_packet_time; //czas przyjscia pakietu do bufora
15     double delivery_time_packet_; //czas dostarczenia pakietu
16     double leave_bufor_packet_time;  //czas opuszczenia pakietu z bufora
17     double delay_packet_timie_;  // opoznienie pakietu
18     double waiting_packet_time; // czas oczekiwania pakietu
19
20
21     Transmitter  *transmiter_ptr;
22     Symulation   *symulation_ptr;
23     Event_list   *event_list_ptr;
24     Medium       *medium_ptr;
25     Source       *src_ptr_;
26 public:
27     Packet();
28     Packet(Transmitter *ptr, Symulation *sym_ptr,Source *source_ptr, double
         time);
29
30     ~Packet();
31
32     double time_in_medium;
33
34
35     void execute(Event_list *list, double clock,Medium *med, Transmitter *ptr,
         Symulation *sym_ptr, double begin_phase_lengt);
36 double current_difs;
37
```

```
38      void set_ptt(int cgp);
39      void inc_ret_number();
40      void set_packet_id(int t, int n);
41      int show_packet_id();
42
43      void set_arrival_packet_time(double time);
44      void set_leave_bufor_packet_time(double time);
45      void set_delivery_packet_time(double time);
46      void set_delay_packet_time(double time);
47      void set_waiting_packet_time(double time);


48
49      double show_leave_bufor_packet_time();
50      double show_arrival_packet_time();
51
52  };
53  #endif // SYMULACJA CYFROWA Packet H
54
```

10.6. Klasa Transmitter

```
1  #include "transmitter.h"
2  #include  "packet.h"
3  #include <iostream>
4
5  using namespace std;
6
7  Transmitter::Transmitter()
8  {
9      //generation_time_ = 10; // tymczasowo
10     negative_transmission_number_ = 0.0;
11     positive_transmission_number_ = 0.0;
12     packet_error_level_ = 0.0;
13 }
14
15
16 Transmitter::~Transmitter()
17 {
18 }
19
20 void Transmitter::generate_packet_(Packet *packet, double time) //funkcja
     generujaca pakiet
21 {
22     ++packet_generation_number;
23     packet->set_packet_id(this->transmiter_id_, this->packet_generation_number);
24     packet->set_arrival_packet_time(time);
25     bufor_.push(packet);
26
27 }
28
29


30 void Transmitter::negative_transmission_number_increment()
31 {
32     negative_transmission_number_++;
33 }
34
35 double Transmitter::show_negative_transmission_number()
36 {
37     return negative_transmission_number_;
38 }
39
```

```
40 void Transmitter::positive_transmission_number_increment()
41 {
42     positive_transmission_number_++;
43 }
44
45 double Transmitter::show_positive_transmission_number()
46 {
47     return positive_transmission_number_;
48 }
49
50 void Transmitter::set_negative_transmission_number(double x)
51 {
52     negative_transmission_number_ = x;
53 }
54
55 void Transmitter::set_positive_transmission_number(double x)
56 {
57     positive_transmission_number_ = x;
58 }
59
60 void Transmitter::set_packet_error_level(double x, double y)
61 {
62     packet_error_level_ = (x / (x + y)) *100;
63 }
64
65 double Transmitter::show_packet_error_level()
66 {
67     return packet_error_level_;
68 }
69
70
```

Plik .h

```
1  #ifndef SYMULACJA_CYFROWA_PACKET_H
2  #define SYMULACJA_CYFROWA_PACKET_H
3  #include "process.h"
4  #include "transmitter.h"
5  class Medium;
6  class Source;
7
8  class Packet:public Process  // klasa Pakiet jest jedynym procesem w tej symulce
9  {
10     double packet_transmission_time_;   // czas transmisji pakietu
11     int retransmission_number_;     // liczba retransmisji
12     int ack_transmission_time_; // czas transmisji ACK  CTIZ =1ms
13     int packet_id;
14     double arrival_packet_time; //czas przyjscia pakietu do bufora
15     double delivery_time_packet_; //czas dostarczenia pakietu
16     double leave_bufor_packet_time;  //czas opuszczenia pakietu z bufora
17     double delay_packet_timie_;  // opoznienie pakietu
18     double waiting_packet_time; // czas oczekiwania pakietu
19
20
21     Transmitter  *transmiter_ptr;
22     Symulation   *symulation_ptr;
23     Event_list   *event_list_ptr;
24     Medium       *medium_ptr;
25     Source       *src_ptr_;
26 public:
27     Packet();
28     Packet(Transmitter *ptr, Symulation *sym_ptr,Source *source_ptr, double
         time);
29
30     ~Packet();
31
```

```
32      double time_in_medium;
33
34
35      void execute(Event_list *list, double clock,Medium *med, Transmitter *ptr,  P
          Symulation *sym_ptr, double begin_phase_lengt);
36  double current_difs;
37
38      void set_ptt(int cgp);
39      void inc_ret_number();
40      void set_packet_id(int t, int n);
41      int show_packet_id();
42
43      void set_arrival_packet_time(double time);
44      void set_leave_bufor_packet_time(double time);
45      void set_delivery_packet_time(double time);
46      void set_delay_packet_time(double time);
47      void set_waiting_packet_time(double time);
48
49       double show_leave_bufor_packet_time();
50       double show_arrival_packet_time();
51
52  };
53  #endif // SYMULACJA CYFROWA Packet H
54
```

10.7. Klasa Process

```
1  #include "process.h"
2  #include  "event.h"
3  #include "packet.h"
4  #include "event_list.h"
5
6  Process::Process()
7  {
8      phase = 0;
9      terminated = false;
10     my_event_ = new Event(this);
11
12 }
13
14 Process::~Process()
15 {
16 }
17
18 void Process::activate(double time, Event_list *list, double clock)
19 {
20     my_event_->event_time_ =  clock + time;
21     list->schedule(my_event_);
22 }
23
24
25
```

Plik .h

```h
#ifndef SYMULACJA_CYFROWA_PROCESS_H
#define SYMULACJA_CYFROWA_PROCESS_H
#include "symulation.h"
class Event;
class Medium;
class Event_list;
class Transmitter;

class Process
{
private:
    Event *my_event_;

public:
    void virtual execute( Event_list *list, double clock, Medium *med,
      Transmitter *ptr, Symulation *sym_ptr, double begin_phase_lengt) {};

    void activate(double time, Event_list *list, double clock);
    int phase;
    bool terminated;
    Process();
    ~Process();


};
#endif // SYMULACJA CYFROWA Process H

```

10.8. Klasa Generator

```cpp
#include "generator.h"
#include <cmath>


Generator::Generator(int kernel): M(2147483647.0), A(16807), Q(127773), R(2836)
{
    kernel_ = kernel;
}

Generator::~Generator()
{
}


double Generator::generator_rownomierny()
{
    int h = int(kernel_ / Q);
    kernel_ = 16807 * (kernel_ -Q * h) - R * h;
    if (kernel_ < 0) kernel_ += M;
    return static_cast<long double>(kernel_) / static_cast<long double>(M);

}


double Generator::generator_wykladniczy(double lambda)
{
    return -log(generator_rownomierny()) / lambda;

}

```

Plik .h

```
#ifndef TEST_LICZB_PSEUDOLOSOWYCH_GENERATOR_H
#define TEST_LICZB_PSEUDOLOSOWYCH_GENERATOR_H
class Generator
{
public:
    Generator(int kernel);
    ~Generator();

    const double M;
    const double A;
    const double Q;
    const double R;
    int kernel_;

    double generator_rownomierny();
    double generator_wykladniczy(double lambda);
};
#endif // SYMULACJA CYFROWA Generator H

```

10.9. Klasa Event

```
#include "event.h"
Event::Event()
{
}
Event::Event(Process* ptr)
{
    event_time_ = -1.0;
    proc_ = ptr;
}
Event::~Event()
{
}
```

Plik .h

```
#ifndef SYMULACJA_CYFROWA_EVENT_H
#define SYMULACJA_CYFROWA_EVENT_H
class Process;

class Event
{
public:
    double event_time_;
    Process *proc_;
    Event();
    Event(Process *ptr);
    ~Event();
};
#endif // SYMULACJA CYFROWA Event H

```

10.10. Klasa Event_list

```
#include "event_list.h"
#include  "link.h"
#include "event.h"
#include <iostream>
#include <windows.h>
using namespace std;


Event_list::Event_list()
{
    head = new Link (new Event());
    number_elements = 0;
}


Event_list::~Event_list()
{
}

void Event_list::schedule(Event*e)      // przeszukiwanie odpowiedniego miejsca na liscie do wstawienia nowego zdarzeia
{

        Link *k;
        for (k = head->prev; k->data->event_time_ > e->event_time_; k = k->prev);
        (new Link(e))->add(k);
        ++number_elements;

}
void Event_list::show_agenda()
{

    Link *p= head->next;
     while(p  != head)
     {
         cout <<endl<< "\t \t event_time: " << p->data->event_time_ << endl <<
           "\t \t *proces : " << p->data->proc_ <<endl<<"\t \t liczba elementow w agendzie: "<<number_elements<< endl;
         cout << "\t \t proces nastepny: " << p->next->data->proc_ << "\t\t
           event time nastepnego procesu: " << p->next->data->event_time_ <<
           endl;
         cout << "\t \t proces poprzedni: " << p->prev->data->proc_ << "\t\t
           event time poprzedniego procesu: " << p->prev->data->event_time_ <<
           endl<<endl;

        p = p->next;
     }
}


bool Event_list::empty()
{
    return head->next == head;
}

Event* Event_list::first()  // pokazuje pierwszy element z listy, poza "sztucznym" zdarzeniem
{

    return head->next->data;
}

```

```
55 Event* Event_list::remove_first(bool show_messages) //usuwanie pierwszego
   elementu z listy
56 {
57     Event *current = first();
58 if (show_messages)  cout <<"current = "<<current<< "\t Usuwany jest pakiet: " <<
   head->next->data<< endl;
59     head->next->remove();
60     --number_elements;
61     current = first();
62 if (show_messages)  cout << "zwracam current: " << current << endl;
63     return current;
64 }
65
66 int Event_list::show_number_elements()
67 {
68     return number_elements;
69 }
70
```

Plik .h

```
1  #ifndef SYMULACJA_CYFROWA_EVENT_LIST_H
2  #define SYMULACJA_CYFROWA_EVENT_LIST_H
3  class Event;
4  class Link;
5
6
7  class Event_list
8  {
9      Link *head;
10     int number_elements;
11 public:
12     Event_list();
13     ~Event_list();
14
15     Event *first();
16     Event *remove_first(bool show_messages);
17     void schedule(Event*e);
18     bool empty();
19
20     void show_agenda();
21     int show_number_elements();
22
23 };
24 #endif // SYMULACJA CYFROWA Event_list H
25
```

10.11. Klasa Link

```cpp
#include "link.h"
Link::Link()
{
}
Link::Link(Event* ptr)
{
    data = ptr;
    next = prev = this;
}
Link::~Link()
{
}

void Link::add(Link* item)  // funkcja wstawia obiekt na liste za elementem wskazywanym przez parametr
{
    prev = item;
    next = item->next;
    item->next = next->prev = this;
}

void Link::remove() // funkcja usuwajaca obiekt z listy
{
    prev->next = next;
    next->prev = prev;
    delete this;
}

```

Plik .h

```cpp
#ifndef SYMULACJA_CYFROWA_LINK_H
#define SYMULACJA_CYFROWA_LINK_H
class Event;
class Event_list;

class Link
{
    Link *next, *prev;
    Event *data;
    friend class Event_list;
public:
    Link();
    Link(Event *ptr);
    ~Link();

    void add(Link *item);
    void remove();


};
#endif // SYMULACJA CYFROWA Link H

```