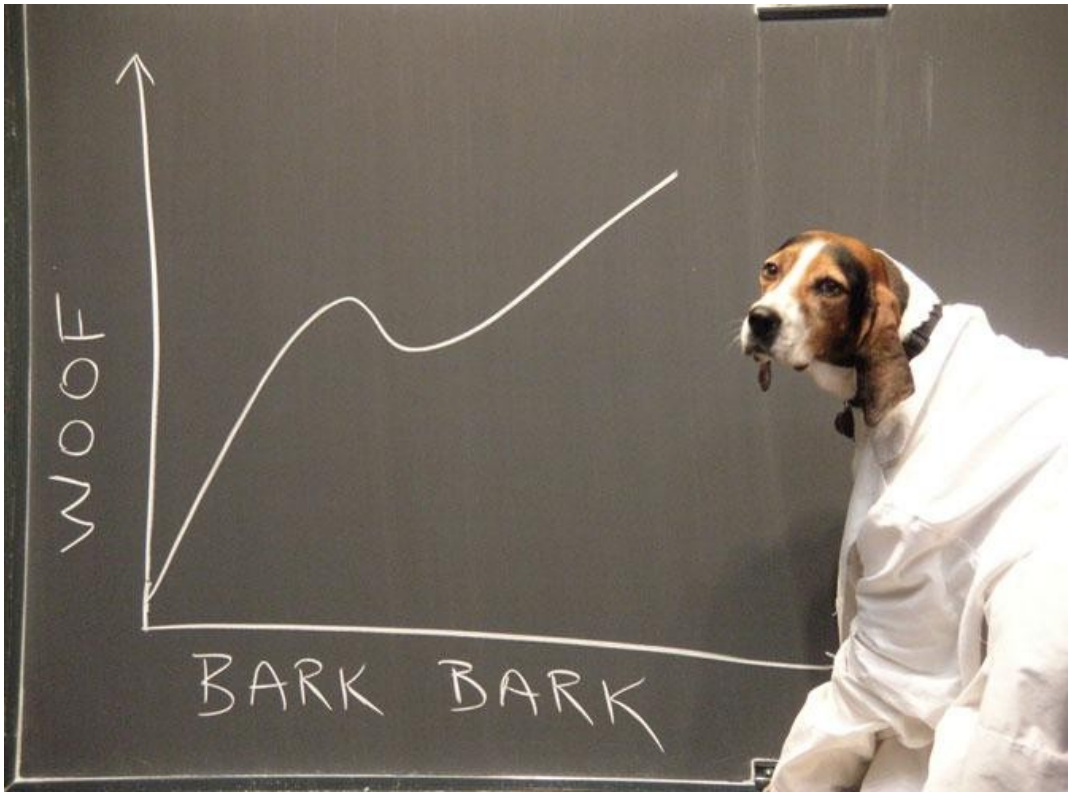
Symulacja Cyfrowa – Projekt

Raport końcowy



Metoda 1 (przeglądanie działań)
A8j - B4 - 2 - 5

1. **Pełny tekst rozwiązywanego zadania.**
W pamięci operacyjnej pewnego systemu komputerowego pojawiają się, w losowych odstępach czasu **TPG,** procesy gotowe do wykonania. Każdy proces żąda dostępu do jednego z **N_P** procesorów na czas **TPW**. System operacyjny wybiera - zgodnie z algorytmem **A** - jeden spośród procesów znajdujących się w stanie gotowości i przydziela mu pierwszy wolny procesor na czas **TPK** ($0 < \mathbf{TPK} \leq \mathbf{TPW}$). Każdy wykonywany proces żąda dostępu do jednego z **N_IO** urządzeń wejścia-wyjścia po czasie **TPIO.** Urządzenia są wybierane z jednakowym prawdopodobieństwem **TPD**. Po zgłoszeniu żądania dostępu do urządzenia wejścia-wyjścia, proces zwalnia procesor i zostaje przeniesiony do jego kolejki przechodząc w stan oczekiwania. Przyznanie nieprzerwanego dostępu do urządzenia wejścia-wyjścia na czas **TPO** odbywa się zgodnie z algorytmem **B** . Po czasie **TPO** proces wraca do stanu gotowości i oczekuje na przydział procesora na czas **TPW** pomniejszony o dotychczasowy czas wykorzystania procesora. Po zakończeniu działania proces przechodzi w stan końcowy i zostaje usunięty z systemu.

   Opracuj symulator planisty przydziału procesora zgodnie z metodą **M.**

   Za pomocą symulacji wyznacz:

   - Wartość parametru **L**, która zapewnia średni czas oczekiwania na procesor, tzn. czas pobytu w kolejce procesów gotowych, nie większy niż 50 ms, a następnie:
     - wykorzystanie procesorów w [%],
     - przepustowość systemu mierzoną liczbą procesów zakończonych w jednostce czasu,
     - średni czas przetwarzania, tzn. czas jaki upływa między zgłoszeniem procesu do systemu, a jego zakończeniem,
     - średni czas odpowiedzi – czas między zgłoszeniem żądania dostępu do jednego z urządzeń wejścia-wyjścia i jego otrzymaniem.

   Sporządź wykres zależności średniego oraz maksymalnego czasu oczekiwania na procesor w zależności od wartości **L**.

   Przyjmujemy następujące parametry:

   TPG - zmienna losowa o rozkładzie wykładniczym o intensywności **L**.

   TPW – zmienna losowa o rozkładzie jednostajnym o wartościach z przedziału <1, 50**>** [ms].

   TPIO – zmienna losowa o rozkładzie jednostajnym o wartościach z przedziału <0, **TPK**-1> [ms], gdzie **TPK** to czas, na który procesor został przydzielony procesowi. Przypadek, w którym wylosowana wartość **TPIO** wynosi 0, interpretujemy jako brak żądania dostępu do urządzenia wejścia-wyjścia.

   TPD – zmienna losowa o rozkładzie jednostajnym o wartościach z przedziału <1, **N_IO>**.

   TPO – zmienna losowa o rozkładzie jednostajnym o wartościach z przedziału <1, 10> ms lub 0, jeśli **TPIO** jest równe 0.

   Zmienna TPG powinna być liczbą rzeczywistą, natomiast pozostałe zmienne liczbami całkowitymi

Algorytm A:

Procesy oczekujące na przydział procesora ustawiane są w dwóch kolejkach: K1 oraz K2. Kolejka K1 jest przeznaczona dla procesów, które dopiero pojawiły się w systemie i nie miały jeszcze przydzielonego procesora. W kolejce K2 znajdują się procesy, które powróciły ze stanu oczekiwania do stanu gotowości.
W momencie zwolnienia jednego z dostępnych procesorów, z prawdopodobieństwem 0.5, zgodnie z algorytmem **A2** jest wybierany proces z kolejki K1 i z prawdopodobieństwem 0.5, zgodnie z algorytmem **A3** jest wybierany proces z kolejki K2.

A2:
Procesor jest przydzielany procesom w kolejności losowej. Każdy proces oczekujący na przydział procesora może uzyskać do niego dostęp z takim samym prawdopodobieństwem, niezależnie od momentu pojawienia się w kolejce. Procesor jest przydzielany procesowi na czas **TPK**=**TPW**.

A3:
Przydział procesora odbywa się metodą SJF (ang. *Shortest Job First*) – najpierw najkrótszy pozostały czas. Wolny procesor jest przydzielany procesowi o najkrótszym czasie **TPW**. W przypadku procesów o tym samym czasie **TPW**, jeden z nich jest wybierany losowo zgodnie z rozkładem jednostajnym. Procesor jest przydzielany procesowi na czas **TPK**=**TPW**.

Algorytm B:

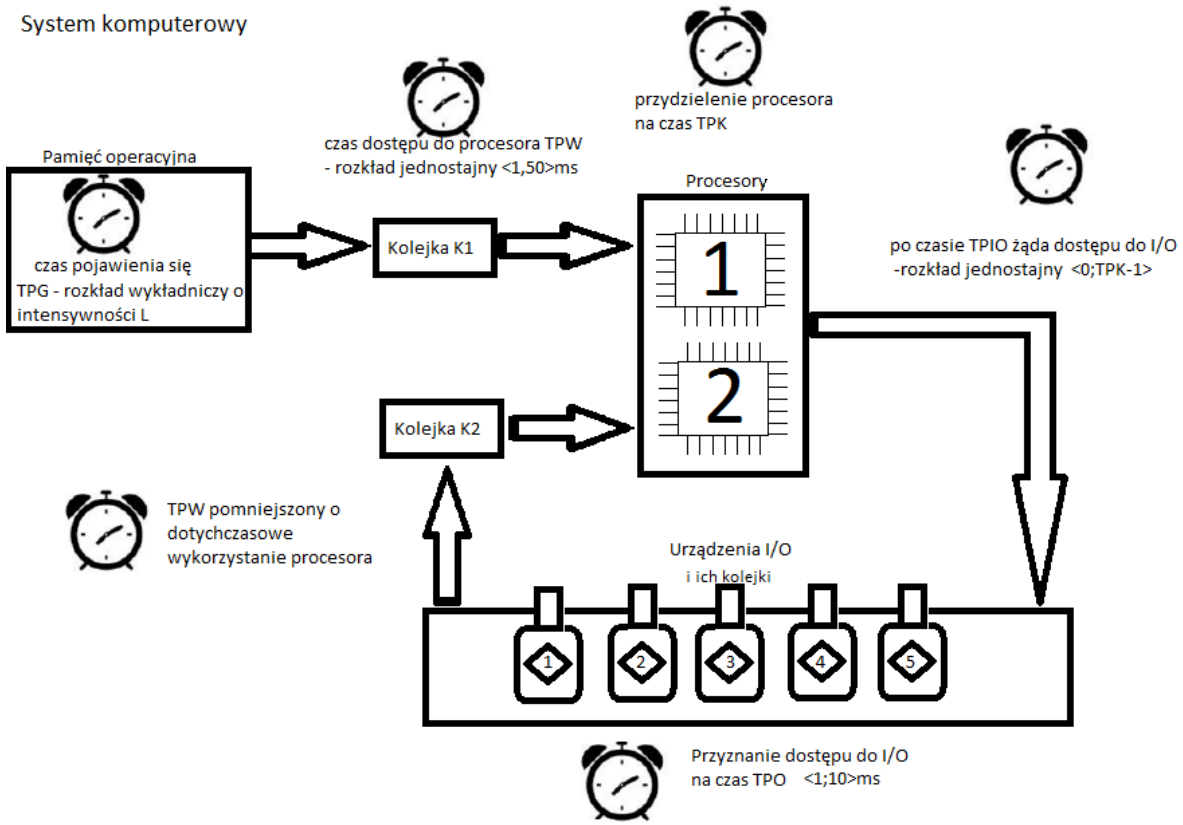
Procesy oczekujące na urządzenie wejścia-wyjścia są obsługiwane zgodnie z metodą priorytetową. Po zwolnieniu urządzenia, jest ono przydzielane procesowi o najwyższym priorytecie na czas **TPO**. W przypadku procesów o tym samym priorytecie, jeden z nich jest wybierany losowo, zgodnie z rozkładem jednostajnym. Priorytet procesu należy wyznaczyć ze wzoru:

$$P = -TPO + TO,$$

gdzie TO jest czasem oczekiwania w kolejce w ms. Po uzyskaniu dostępu do urządzenia, czas TO powinien zostać wyzerowany.

2. Krótki opis modelu symulacyjnego.

Schemat modelu Symulacyjnego



Opis obiektów i ich atrybutów

Obiekt	Nazwa klasy implementującej obiekt	Opis	Atrybuty
System komputerowy	*SystemKomputerowy*	System komputerowy jest klasą wdrażającą nowe procesy do naszego systemu. To tutaj tworzymy kolejki z których następnie procesor będzie wybierał procesy do obsługi (kolejka typu *Kolejka* oraz *SJF*).	-tablica kolejek typu *SJF.*
Proces	*Proces*	Klasa reprezentuje procesy które pojawiają się w systemie komputerowym. Procesy pojawiają się w losowych odstępach czasu **TPG** i zostają umieszczone w kolejce *K1*. Każdy proces żąda dostępu do jednego z procesorów na czas **TPW**. Po czasie **TPIO** zgłasza żądanie dostępu do urządzenia *IO* (o ile TPIO nie jest '0'), Zwalnia procesor i zostaje przeniesiony do kolejki urządzenia a następnie do samego urządzenia na czas **TPO,** po którym trafia do kolejki *SJF* i czeka na	-czas dostępu do procesora *TPW* typu *int.* -czas żądania dostępu do urządzenia wej-wyj *TPIO* typu *int.* -czas dostępu do urządzenia wej-wyj *TPO* typu *int.* -priorytet dla IO typu *int.* -czas stworzenia procesu

		przydzielenie do procesora.	typu *double.* -czas zgłoszenia dostępu do IO typu *double.* -czas czekania w kolejce typu *double.* -czas uzyskania dostępu do procesora typu *double.*
Procesor	*Procesor*	Klasa reprezentuje pojedynczy procesor. Procesory są zajmowane przez procesy na czas **TPK**(=TPW) i zwalniane w momencie prośby dostępu do IO lub końca obsługi.	-Wskaźnik na obsługiwany proces typu *Proces*.*
Urządzenie wej/wyj	*IO*	Klasa reprezentuje urządzenia wejścia-wyjścia. Urządzenie jest zajmowane przez proces na czas **TPO.** Proces trafia do urządzenia z jego kolejki priorytetowej *KolejkaPrio.*	-Wskaźnik na obsługiwany proces typu *Proces*.* -Wskaźnik na kolejkę priorytetową typu *KolejkaPrio*.*
Kolejka Priorytetowa	*KolejkaPrio*	Kolejka do której trafiają procesy zwolnione przez procesor z powodu żądania dostępu do urządzeń wej-wyj. Procesy są w niej poukładane w zależności od ich *priorytetu.*	-Wskaźnik na listę jednokierunkową typu *Pole*.* -zmienna określająca wielkość kolejki typu *int.*
Kolejka SJF	*SJF*	Kolejka do której trafiają procesy zwolnione z urządzeń I/O. Procesy poukładane są zgodnie z metodą SJF -najpierw najkrótszy pozostały czas. Dziedziczy po klasie *KolejkaPrio.*	-Dziedziczy atrybuty po *KolejkaPrio.*
Kolejka	*Kolejka*	Kolejka do której trafiają procesy które dopiero pojawiły się w systemie. Dziedziczy po klasie *SJF.*	-Lista procesów typu *dequeue<Proces*>*
Pole	*Pole*	Element zaimplementowanej listy jednokierunkowej używanej przez klasie *KolejkaPrio* i *SJF.* Zaprzyjaźniona z klasami *SJF* i *KolejkaPrio*.	-Wskaźnik na następne pole typu *Pole*.* -Wskaźnik na obecny proces typu *Proces*.*
Model symulacji	*Model*	W tej klasie tworzymy wszystkie obiekty niezbędne do symulacji, to tutaj znajduję się główna pętla programu wraz z obsługą zdarzeń.	-Tablica wskaźników typu *IO.* -Tablica wskaźników typu *Procesor.* -Wskaźnik na system komputerowy typu *SystemKomputerowy.*

			-Wskaźniki na Zdarzenia typu: *NowyProces, ProsbaDostepuIO, ZakonczenieObslugiIO, WykonczProces, WolnyProcesor, DostepDoIO.*
Generator liczb	*Random*	Klasa służy do generowania liczb pseudolosowych. Używa Generatora Multiplikatywnego.	-jądro generatora typu *static int64_t.* -stała liczba pierwsza typu *static int64_t.* -Współczynnik dla rozkładu wykładniczego typu *static double.*

Opis zdarzeń czasowych i warunkowych.

Zdarzenia czasowe:

-pojawienie się procesu.
-żądanie dostępu do urządzenia I/O.
-zakończenie obsługi urządzenia I/O.
-usuniecie zużytych procesów.

Zdarzenia warunkowe:

-przydzielenie procesora.
-przyznanie nieprzerwanego dostępu do urządzenia I/O.

Szczegółowy opis zdarzeń czasowych

Zdarzenie	Opis	Algorytm
Pojawienie się procesu	Zdarzenie generowane przez źródło *TPG* oznaczające pojawienie się nowego procesu w systemie, po czym planujemy pojawienie się kolejnego nowego procesu.	1. Stwórz proces. 2. Umieść proces w kolejce K1. 3. Zaplanuj dodanie następnego procesu.
Żądanie dostępu do urządzenia I/O	Zdarzenie generowane przez czas *TPIO(!=0)* oznaczające zwolnienie procesu przez procesor oraz umieszczenie go w kolejce do urządzenia I/O. (jeśli *TPIO* == 0, to zdarzenie nie wystąpi)	1. Zwolnij Procesor. 2. Dodaj proces do kolejki urządzenia I/O.
Zakończenie obsługi urządzenia I/O	Zdarzenie generowane czasem *TPO* zwalnia urządzenie I/O i kieruje jego proces do kolejki *SJF.*	1. Zwalniamy urządzenie I/O. 2. Dodajemy proces do kolejki *SJF.*
Usunięcie zużytych procesów	Zdarzenie oznacza bezpowrotne usunięcie procesu systemu. Generowane wykorzystaniem całego czasu *TPW.*	1. Zwolnij procesor i usuń proces z systemu.

Szczegółowy opis zdarzeń warunkowych

Zdarzenie	Opis	Algorytm
Przydzielenie procesora	Kolejka z której zostanie wylosowany proces wybrana jest z jednakowym prawdopodobieństwem, a wybrany proces zostaje umieszczony w obsłudze procesora.	Jeśli jedna z kolejek nie jest pusta a procesor jest wolny: 1. Wybieramy kolejkę i zgodnie z algorytmem kolejki pobieramy proces(usuwamy z kolejki). 2. Przydzielamy proces pierwszemu wolnemu procesorowi. 3. Planujemy zdarzenie Żądanie dostępu do IO.
Przyznanie nieprzerwanego dostępu do urządzenia I/O	Pobieramy proces z kolejki urządzenia i zaczynamy jego obsługę na czas *TPO.*	Jeśli urządzenia są wolne i proces oczekuje w kolejce urządzenia: 1. Aktualizujemy priorytety i wybieramy proces z kolejki. 2. Umieszczamy proces w obsłudze urządzenia. Gdy *TPIO != 0.* 3. Planujemy zdarzenie Zakończenia obsługi IO. Gdy *TPIO == 0.* 3.Planujemy zdarzenie Usunięcia zużytych procesów.

3. Lista parametrów wywołania programu symulacyjnego.

Kernel – parametr oznaczający jądro naszego generatora liczb pseudolosowych.

Intensywność L – parametr oznaczający intensywność naszego generatora liczb pseudolosowych o rozkładzie wykładniczym.

Liczba iteracji dla procesów – określa ilość rejestrowanych wyników zależnych od liczby procesów.

Czas Symulacji – określa czas rejestrowania wyników.

Stacjonarność – liczba procesów po których zakończeniu mija faza przejściowa.

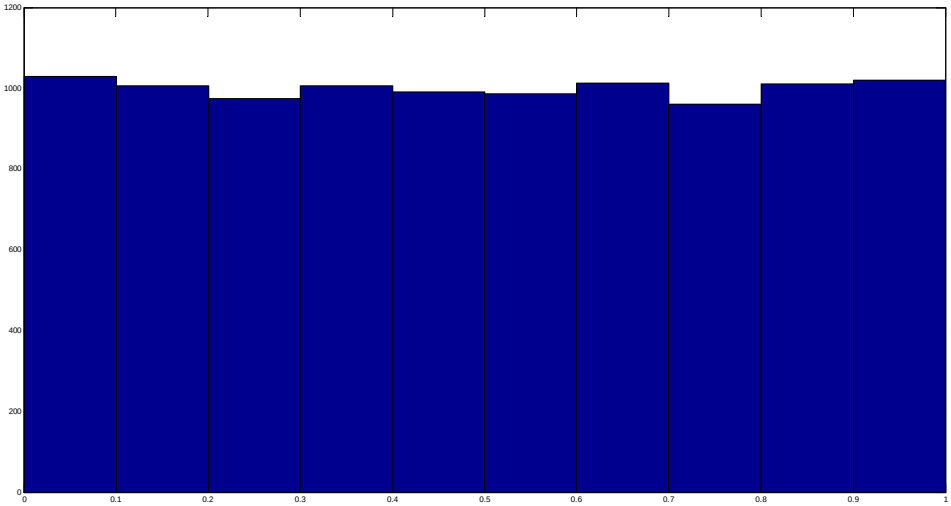
4. Krótki opis zastosowanych generatorów liczb losowych, ich realizacji programowej:

Zastosowałem następujące generatory liczb pseudolosowych:

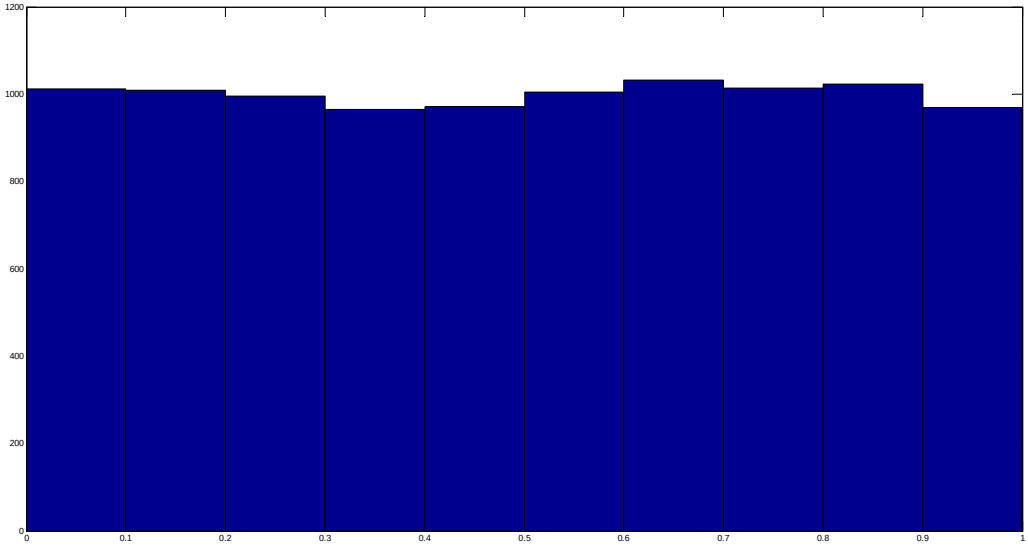
1. Generator multiplikatywny dla rozkładu równomiernego.
   Invocom, Generatory, strona 6.

   Histogramy 10 000 próbek.
   (Generatory wygenerowane poczynając od bazowego kernel = 1117, kolejne ziarna przyjęte jako 200 000 wywołanie poprzedniego generatora):

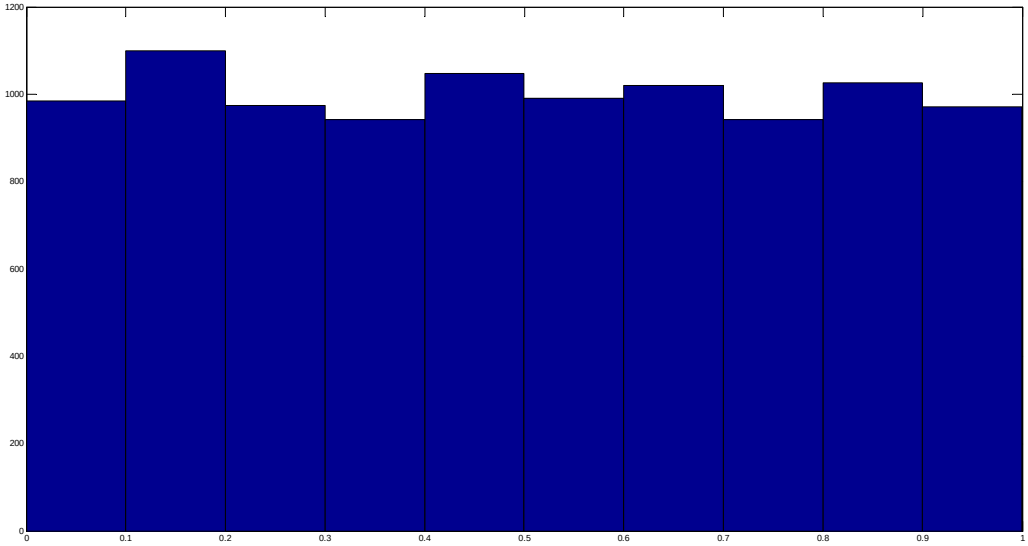
$$\mu = 0.4998 \qquad \sigma = 0.0839$$



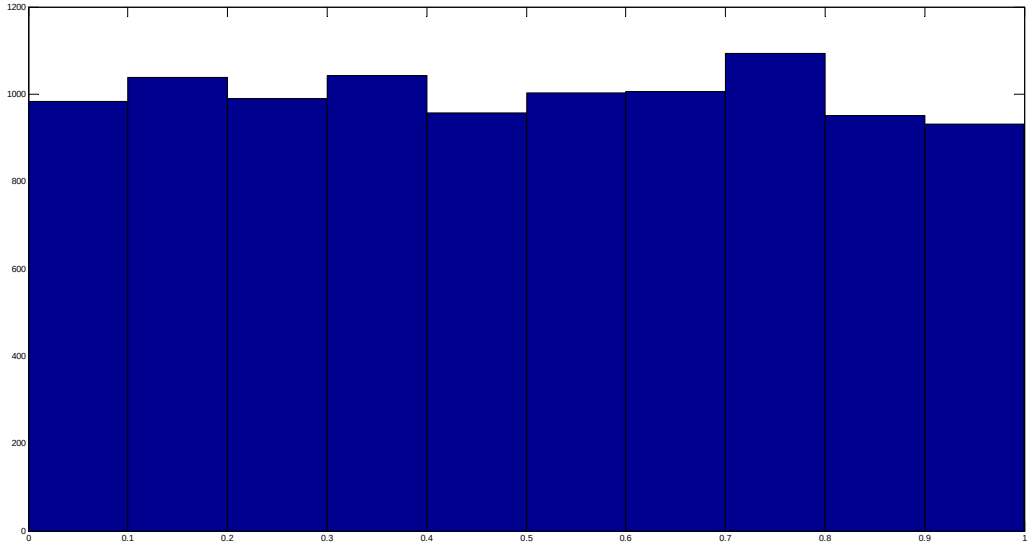
$\mu = 0.5001 \qquad \sigma = 0.0835$



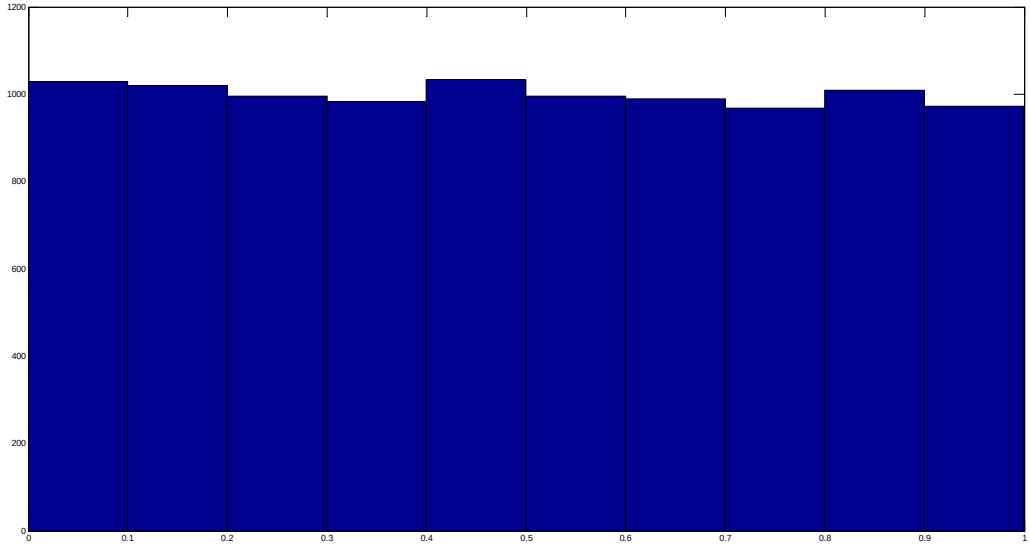
$\mu = 0.4967 \qquad \sigma = 0.0832$



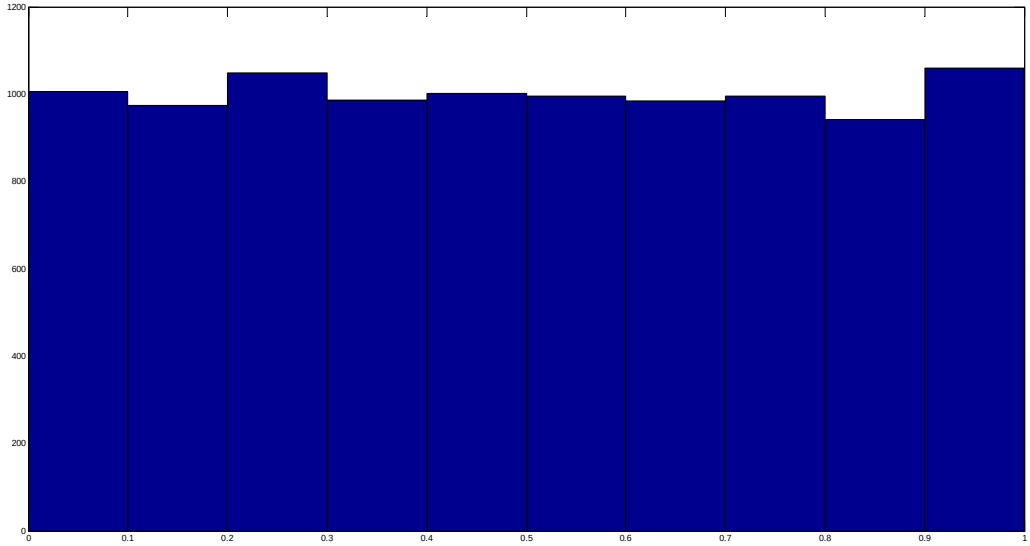
$\mu = 0.4965 \qquad \sigma = 0.0820$



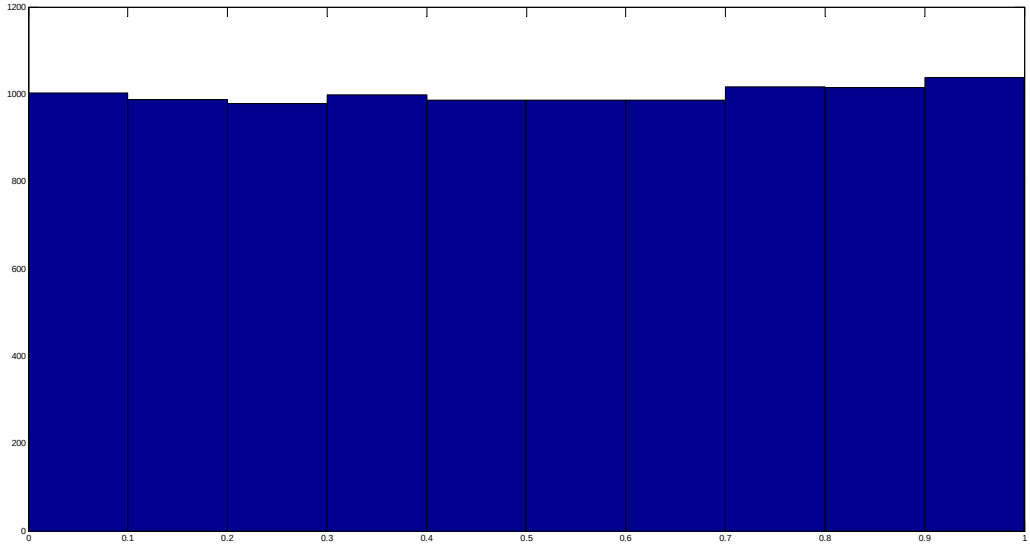
$\mu = 0.4963 \qquad \sigma = 0.0832$



$\mu = 0.4999 \qquad \sigma = 0.0840$



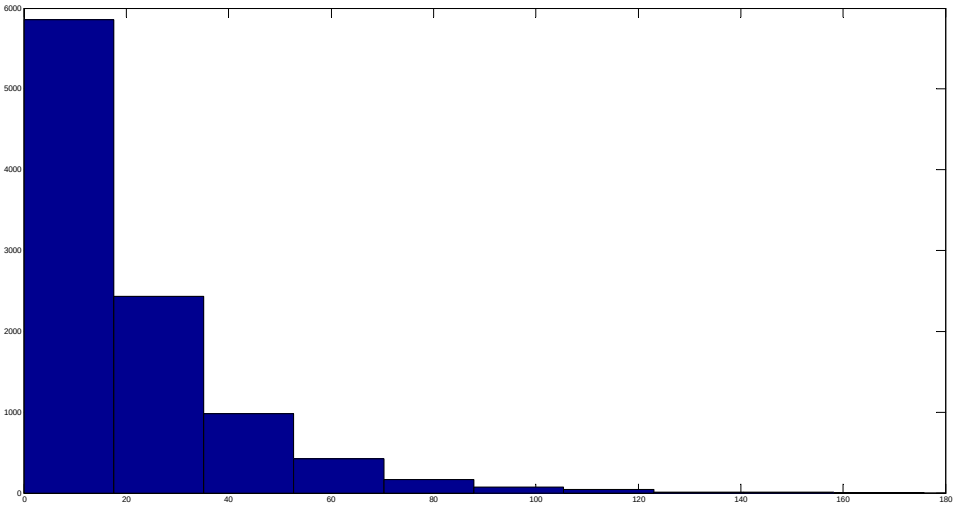
$\mu = 0.5031 \qquad \sigma = 0.0839$



2. Generator o rozkładzie wykładniczym zbudowałem na zasadzie odwrotnej dystrybuanty z generatora o rozkładzie równomiernym.
   Invocom, Generatory, strona 13.

   Histogram 10 000 próbek(L = 0.05)

$$\mu = 19.9001 \qquad \sigma = 403.7346$$



3. Wyjaśnienie, w jaki sposób została zapewniona niezależność sekwencji pseudolosowych w kolejnych przebiegach symulacji:

   Stosuje metodę replikacji – powtarzam kilka razy dla niezależnych przebiegów, czyli z różnymi jądrami generatorów liczb pseudolosowych co pozwala mi zapewnić niezależność sekwencji w kolejnych przebiegach symulacji. Każde kolejne ziarno generatora jest sugerowane przez algorytm wyznaczany jako 200 000 ziarno uzyskane z poprzedniego generatora, podobnie jest z ziarnem dla następnej symulacji.

5. **Opis zastosowanej metody testowania i weryfikacji poprawności działania programu.**
   1. Zdefiniowanie warunków/kryteriów określających poprawną pracę programu:

      - Dobór odpowiedniego parametru L, tak aby średni czas oczekiwania na procesor był nie większy niż 50ms,

      - Odpowiednio duża liczba zakończonych procesów oraz czas trwania programu głównego zapewniający wiarygodność danych losowych.

      - Wyniki poszczególnych symulacji nie mogą od siebie znacznie odbiegać.

   2. Opisanie kroków podjętych w celu stwierdzenia czy program działa poprawnie czy też nie:

      - Kompilacja i uruchomienie programu.

      - Sprawdzenie czy program nie wypisuje błędnych danych (NaN, dzielenie przez 0).

      - Znalezienie momentu stabilizacji systemu.

      - Stwierdzenie, czy przy optymalnym wyborze argumentów wejściowych systemu procesory nie ulegają zapychaniu.

      - Wykonuje kilka symulacji z różnymi parametrami L oraz liczbą iteracji.

      - Wyciągam wnioski z otrzymanych danych i wdrażam poprawki mające na celu uzyskanie średniego czasu oczekiwania nie większy niż 50ms.

      - Uśredniając wyniki z kilku symulacji otrzymuje wiarygodne wyniki na temat stabilności systemu oraz odpowiednich danych wejściowych.

   3. Oszacowanie zakresu wartości parametrów programu, przy których program będzie poprawnie pracował:

      - Korzystając z metody bisekcji doszedłem do następujących parametrów wymaganych do uzyskania dalszych wyników:

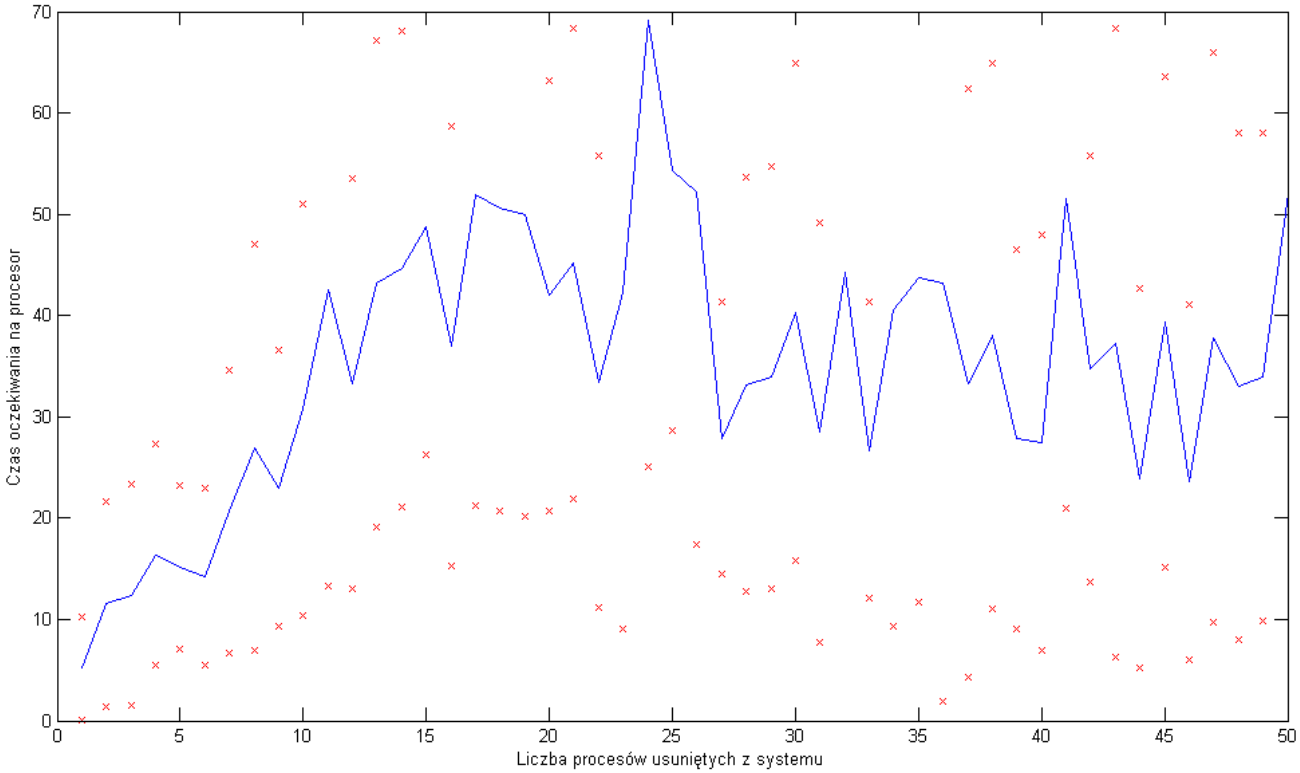
      - liczba przebiegów symulacji: *20*.

      - liczba zakończonych procesów wymagana do uzyskania wyników: *~50 000.*

      - Czas trwania obliczeń: *~200 000.*

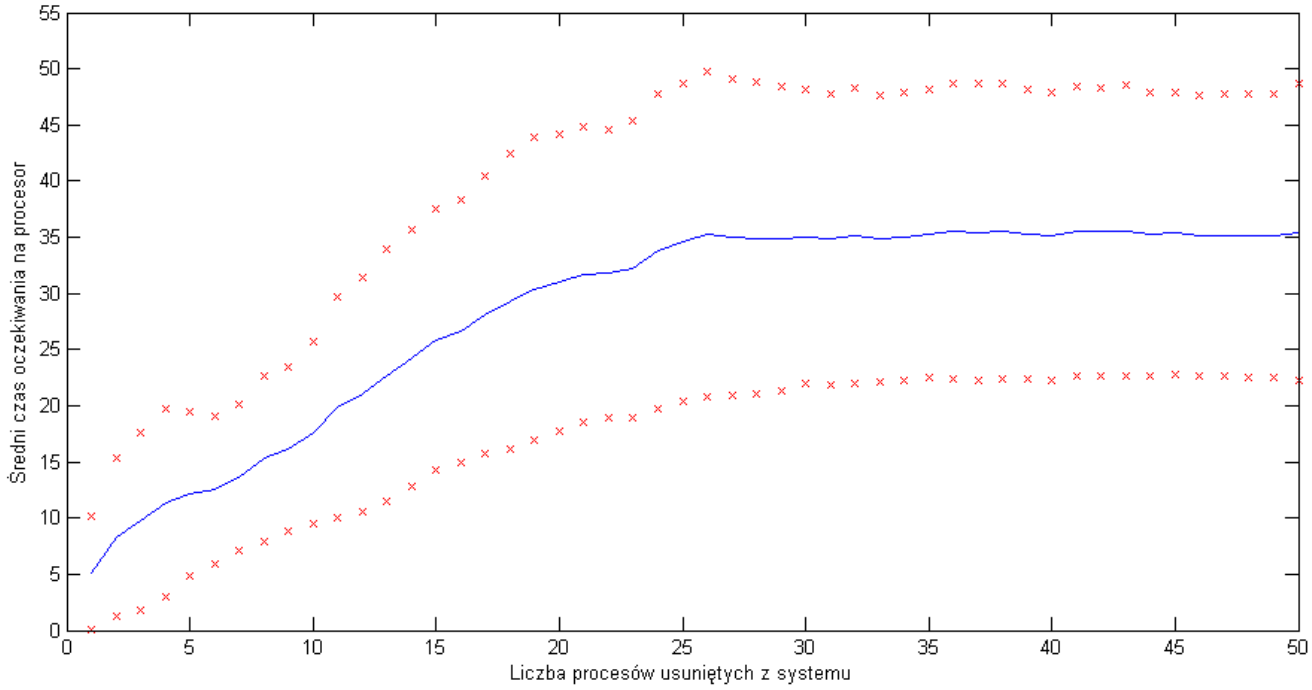
      - Wartość L gwarantująca średni czas oczekiwania nie większy niż 50ms badana w zakresie: *< 0.06 ; 0.07 >*

6. **Opis czynności wykonanych w ramach przygotowania do przeprowadzenia symulacji, plan symulacji.**
   1. Założenia:
      - Warunek końca symulacji: określona z góry liczba zakończonych procesów oraz czas symulacji.
   2. Wstępne symulacje:
      - Ustalenie czasu trwania fazy przejściowej dla następujących parametrów wejściowych:
        - ✗ L=0.06
        - ✗ Kernel = 1129
        - ✗ Dla wartości chwilowej czasu oczekiwania na procesor:



- ✗ można dostrzec faze przejściową trwającą około 25 procesów mimo iż przedział ufności(zaznaczony na czerwono) nie jest w pełni satysfakcjonujący.

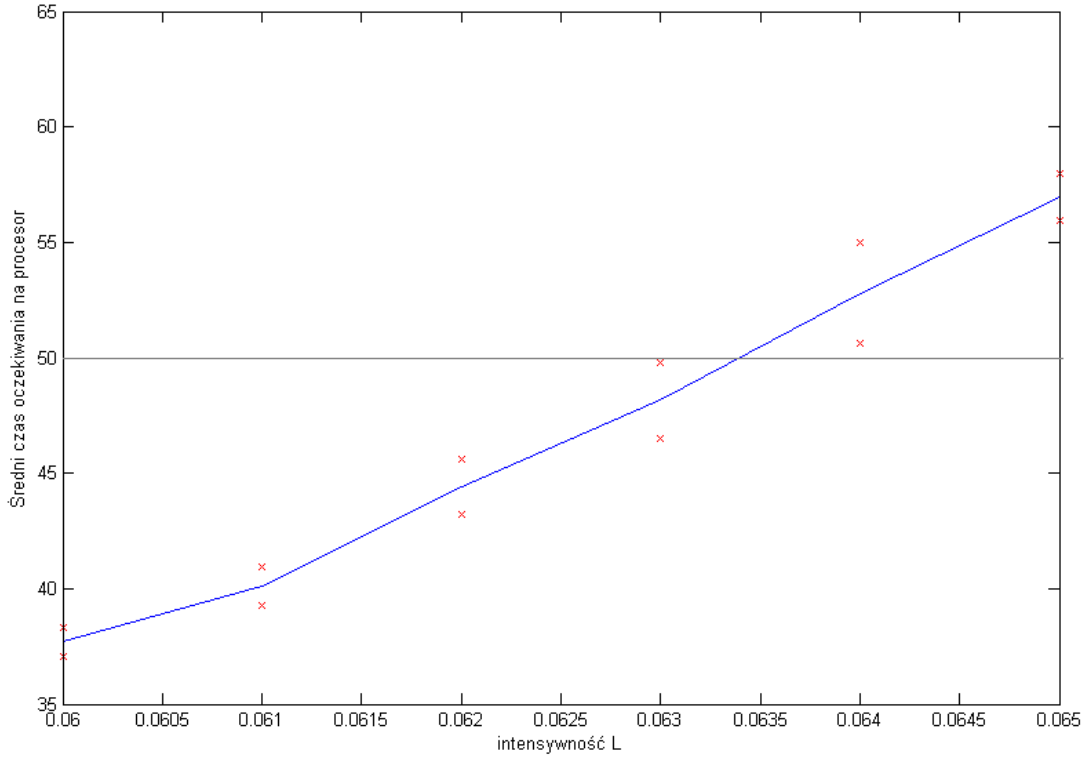
- Dla uśrednionego czasu oczekiwania:



- Faza przejściowa również trwa około 25 procesów, wyniki bardzo wiarygodne ze względu na stabilny przedział ufności(zaznaczony na czerwono).

L	Średnia	Odchylenie standardowe	Przedział ufności
0,06	37,6939	0,8778	0,6279
0,061	40,1092	1,1828	0,8462
0,062	44,4109	1,6310	1,1668
0,063	48,1701	2,3052	1,6490
0,064	52,8024	3,0520	2,1833
0,065	56,9577	2,3262	1,6641
0,066	64,7783	2,6712	1,9109
0,067	70,0968	3,7738	2,6996
0,068	78,2126	5,4431	3,8937
0,069	87,3826	7,4400	5,3223
0,07	98,4089	5,5109	3,9423
0,071	114,6816	10,4578	7,4811
0,072	128,8217	10,3458	7,4010
0,073	149,4382	15,5287	11,1086
0,074	182,6112	12,7460	9,1179
0,075	222,7016	43,1409	30,8611
0,076	296,7574	57,0172	40,7876
0,077	424,2750	174,1478	124,5778
0,078	630,6946	287,1134	205,3886
0,079	1077,8370	369,9465	264,6438

- Ustalenie optymalnej wartości parametru intensywności L w zależności od średniego czasu oczekiwania na procesor.



  - ✗ Wykonano po 10 symulacji dla L z zakresu przedstawionego w tabeli, liczbie zakończonych procesów = 50 000, Czasowi symulacji = 200 000 oraz bazowemu ziarnie: *kernel = 1129*.

3. Wnioski z wstępnych symulacji:

   - Analizując otrzymane wyniki można przyjąć czas trwania fazy przejściowej jako 25 procesów

   - Zgodnie z wynikami i bardzo dobrym przedziałem ufności bez wątpienia przyjmujemy *L = 0.063*

7. Wyniki symulacji.

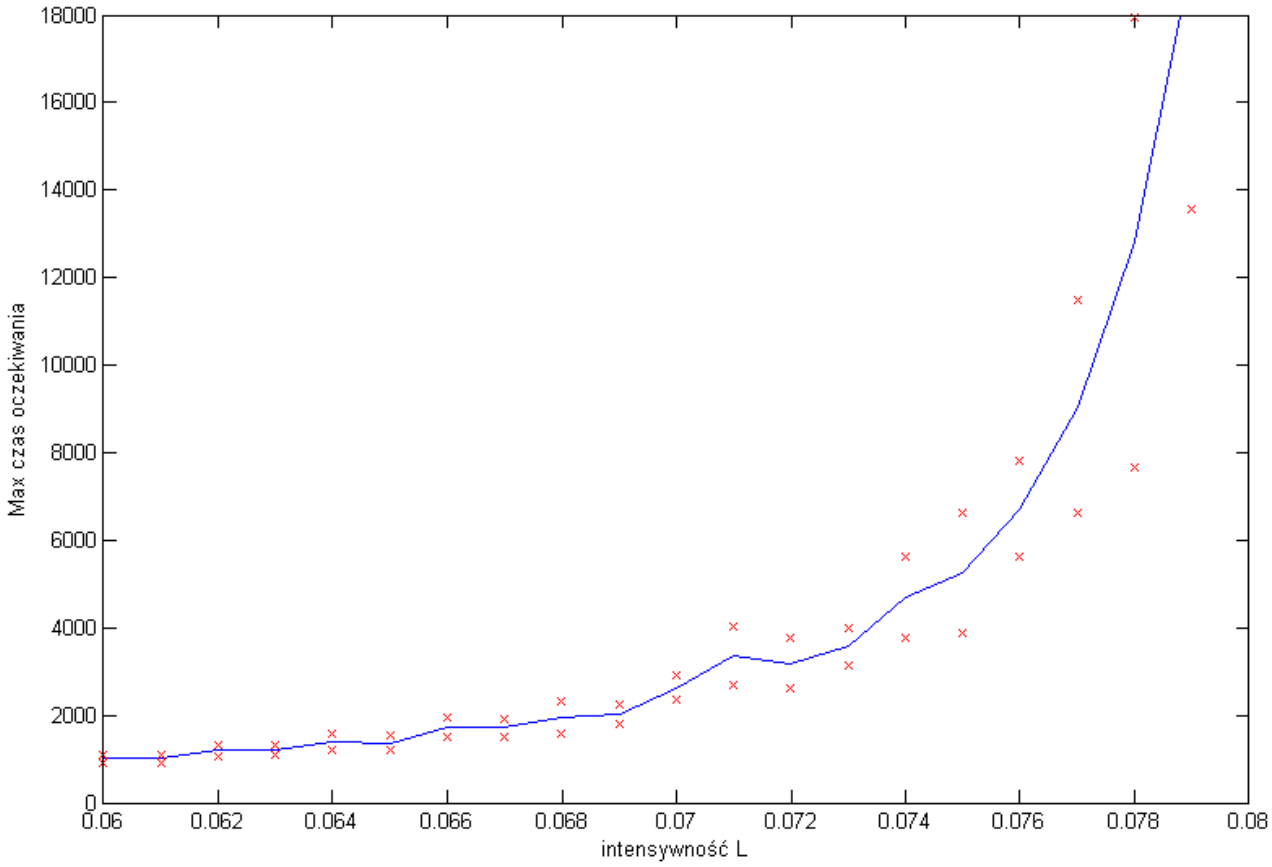
1. Tabela z wynikami opracowanymi za pomocą LibreOfficeCalc:

nr symulacji	Maksymalny czas oczekiwania na procesor	Średni czas oczekiwania na procesor	Wykorzystanie procesora nr 0	Wykorzystanie procesora nr 1	Przepustowość	Średni czas przetwarzania	Średni czas odpowiedzi
1	1226,6874	46,4234	82,9756	73,9780	0,0625	89,5157	0,7494
2	1436,0298	46,2967	83,1940	74,6082	0,0628	88,4939	0,7864
3	1261,0330	51,4386	82,9243	74,4449	0,0633	85,2113	0,7668
4	987,2616	51,1601	83,4705	74,9194	0,0632	91,5606	0,7664
5	1067,6662	46,9756	82,9415	74,2718	0,0629	84,6774	0,7938
6	1062,7153	47,0376	82,6944	74,2767	0,0629	88,7379	0,7516
7	1117,9803	46,1265	83,1964	74,8030	0,0628	86,9853	0,7747
8	1393,7023	51,4563	83,2452	74,5845	0,0629	88,8163	0,7588
9	1237,6069	48,5655	83,3943	75,2167	0,0629	90,5097	0,7672
10	1204,5374	46,2210	82,8311	74,2476	0,0625	87,8903	0,7596
11	1300,5169	50,8217	82,7575	74,1868	0,0633	90,4955	0,7908
12	1246,4602	47,2513	83,0115	74,4041	0,0631	86,1050	0,7842
13	1413,0301	49,5540	81,9534	72,5110	0,0629	81,2358	0,7678
14	1219,0365	45,2584	83,0008	74,5508	0,0629	86,3788	0,7780
15	1134,5658	49,9669	83,7407	75,3836	0,0632	94,5061	0,7873
16	984,8165	49,2472	83,1064	74,8072	0,0631	88,5860	0,7734
17	1193,5804	48,2270	82,7131	73,9201	0,0633	85,0215	0,7970
18	1482,8900	51,8177	83,1531	74,9697	0,0630	90,1339	0,7887
19	1156,7353	45,8437	82,5807	74,0865	0,0626	88,1509	0,7536
20	1287,7969	50,2046	83,1004	74,7724	0,0633	87,2023	0,7862
średnia:	**1220,7324**	**48,4947**	**82,9992**	**74,4471**	**0,0630**	**88,0107**	**0,7741**
odchylenie standardowe:	**137,3118**	**2,1358**	**0,3641**	**0,5886**	**0,0002**	**2,8066**	**0,0144**
Przedział ufności:	**64,2639**	**0,9996**	**0,1704**	**0,2755**	**0,0001**	**1,3135**	**0,0068**

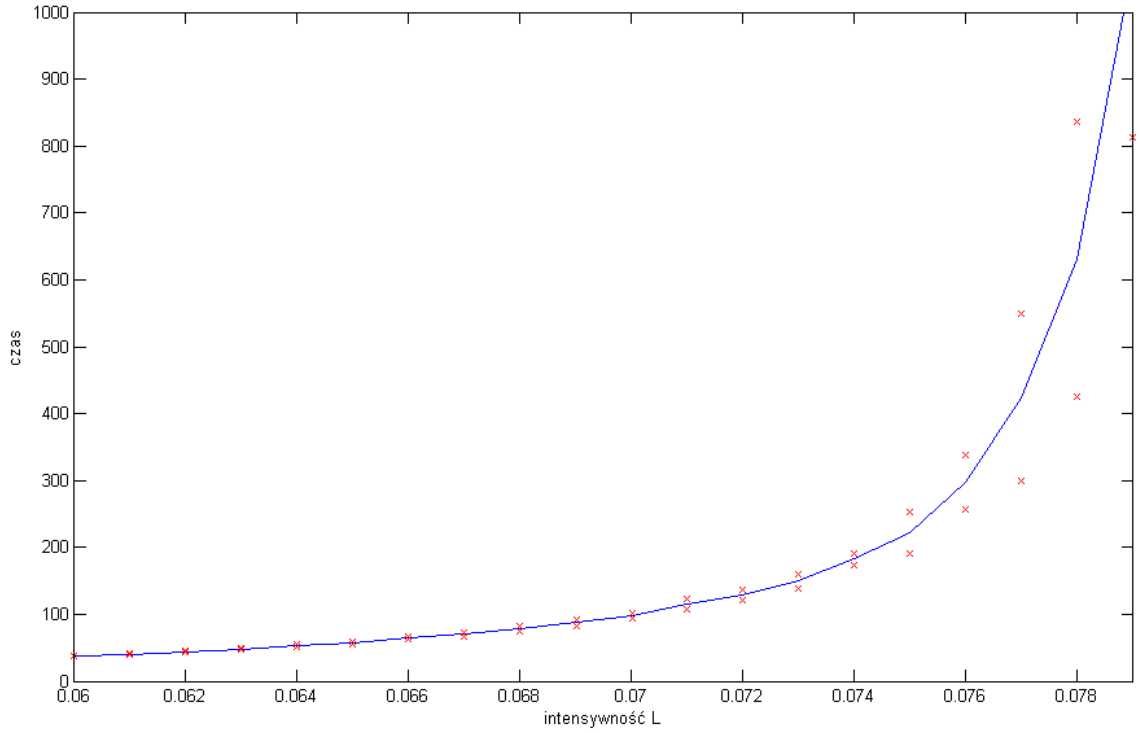
2. Symulacje zostały wykonane z parametrami **L = 0.063** dla 50 000 procesów oraz czasowi symulacji równemu 200 000.
   - Jako ziarno generatora przyjęto 1129.
   - Przedział ufności **α = 95%** dla rozkładu t-Studenta.

3. Badanie średniego oraz maksymalnego czasu oczekiwania na procesor w zależności od wartości L.
   - Maksymalny czas oczekiwania, uzyskany dla 10 symulacji, *kernel = 1129*.
   - L❷€ < 0.060 0.079 >

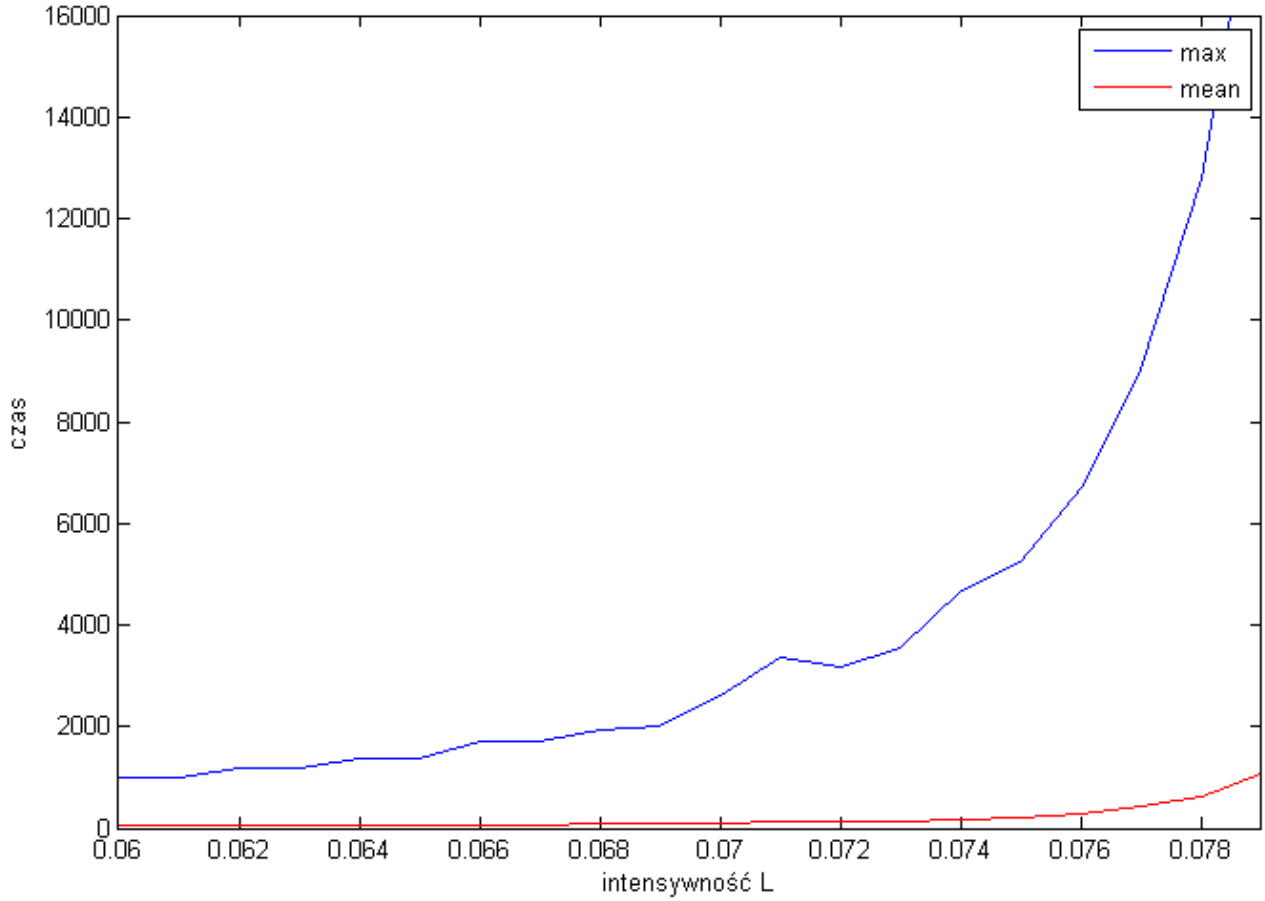
max	Odchylenie standardowe	Przedział ufności
1002,5702	106,7231	76,3451
1010,8686	130,0602	93,0394
1196,0808	181,2308	129,6447
1199,5220	144,3163	103,2376
1380,3376	251,1332	179,6499
1369,8509	227,2109	162,5369
1723,3737	295,2002	211,1735
1715,5082	282,5422	202,1185
1944,2583	538,7121	385,3714
2026,0781	320,9586	229,5999
2628,5486	378,8010	270,9779
3353,0224	941,1608	673,2659
3183,3857	807,3548	577,5469
3564,9010	612,1708	437,9206
4689,1632	1284,6398	918,9760
5253,5975	1907,4295	1364,4928
6713,4806	1531,1283	1095,3032
9050,3725	3393,9778	2427,9054
12807,4033	7203,0004	5152,7161
19328,3465	8075,4049	5776,7967



- Średni czas oczekiwania(tabela pkt 6.2).



- Średni oraz maksymalny czas oczekiwania w zależności od L.



8. Dyskusja i interpretacja otrzymanych wyników.

1. Wnioski.

- Symulacja wymaga sporej ilości iteracji ponieważ procesy pojawiają się losowo w czasie o rozkładzie wykładniczym, co powoduje znaczne, ale rzadkie fluktuacje które należy uśrednić.

- Łatwo można zauważyć, że wykorzystując procesor w ~80% osiągamy optymalne wyniki dla naszej symulacji. Jest to spowodowane odpowiednio dobranym parametrem L który jest odpowiedzialny za intensywność rozkładu wykładniczego zgodnie z którym pojawiają się nowe procesy, a kolejki do procesorów nie są zapchane.

- Przepustowość systemu która jest mierzona liczbą procesów zakończonych w jednostce czasu jest najmniej fluktuującym się parametrem względem innych symulacji co świadczy o poprawnym działaniu symulatora.

- Średni czas przetwarzania, tzn. czas jaki upływa między zgłoszeniem procesu do systemu, a jego zakończeniem również nie odbiega znacznie od normy, lecz nie jest już tak jednolity jak przepustowość systemu i zapewne jest to spowodowane możliwością zakończenia procesu bez zgłoszenia go do kolejek urządzeń wejścia - wyjścia.

- Średni czas odpowiedzi, czyli czas między zgłoszeniem żądania dostępu do jednego z urządzeń wejścia – wyjścia i jego otrzymaniem charakteryzuje się dużą stabilnością w różnych przebiegach symulacji, jest to zrozumiałe. Dzieje się tak z powodu znacznej ilości urządzeń wejścia – wyjścia które nie spotykają się z problemem zapchanych kolejek.

- Średni czas oczekiwania na procesor, tzn czas pobytu w kolejce procesów gotowych potrzebuje dużej liczby iteracji by ustabilizować swój uśredniony przebieg, jest spowodowane pojawianiem się procesów w losowych odstępach czasu TPG - zmiennej losowej o rozkładzie wykładniczym i intensywności L.

- Zależności średniego oraz maksymalnego czasu oczekiwania mają słuszną właściwość – wartości dla różnych symulacji w zależności od intensywności L są do siebie proporcjonalne i rosną wykładniczo wraz z wzrostem wartości L, jest to logiczne biorąc pod uwagę zmienną losową o rozkładzie wykładniczym która generuje procesy, gdy będą się pojawiać zbyt często to ugrzęzną w kolejkach do naszych dwóch procesorów które nie są wstanie obsłużyć tak licznej liczby zgłoszeń.

2. Propozycje ulepszeń.

   - Główny problem z duża ilością czasu oraz procesów wymaganych do poprawnej symulacji spowodowany jest rozkładem wykładniczym czasu pojawiania się nowych procesów, przy zmianie tego czasu na zmienną losową o rozkładzie jednostajnym drastycznie zmniejszylibyśmy rozrzut otrzymywanych danych, a także procesory mogłyby dzięki temu pracować wydajniej.

   - Liczba procesorów w pewien sposób ogranicza możliwości naszej symulacji, zwiększenie ich liczby spowodowałoby znaczny wzrost przepustowości i redukcje czasu oczekiwania na procesor.

   - Dużym usprawnieniem byłoby rozdzielenie procesorów na obsługujące nowe procesy i powracające z obsługi urządzenia wejścia-wyjścia, wtedy procesy sprawniej opuszczałyby system komputerowy, oczywiście w przypadku gdy jeden z procesorów jest bezczynny mógłby nadal obsłużyć „nie swoją" kolejkę.

   - Podsumowując największą wadą systemu jest wydajność procesorów i ich nie radzenie sobie z dużym nierównomiernym napływem procesów, a więc najtrafniejszym rozwiązaniem byłoby zwiększenie ilości procesorów a także zmiana sposobu zarządzania nimi.

9. Kod programu.

dostep_do_io.h

```
#ifndef SYMULACJA_DOSTEP_DO_IO_H_
#define SYMULACJA_DOSTEP_DO_IO_H_

class ZakonczenieObslugiIO;
class IO;

//Jesli urzadzenia sa nieobslugiwane, to sprawdzamy czy proces czeka w kolejce na obsluge
//Nastepnie proces opuszcza kolejke urzadzenia i zaczyna jego obsluge na czas TPO <1:10>[ms]
//Procesy sa wybierane w kolejnosci od najwyzszego priorytetu
//w konstruktorze podajemy zakonczenie_obslugi_io by moc zmodyfikowac czas nastepnego
zdarzenia
class DostepDoIO
{
public:
 DostepDoIO(IO** io, ZakonczenieObslugiIO* zakonczenie_obslugi_io);
 void Wypisz(int i);
 //przydziela proces z kolejki na czas TPO
//proces przydzielany zgodnie z priorytetem
//ktory jest uaktualniony przed przydzialem
 //jako argument i podajemy numer urzadzenia io
 void Wykonaj(int i, int iteracje);
private:
 IO** io_;
 ZakonczenieObslugiIO* zakonczenie_obslugi_io_;
};
#endif
```

dostep_do_io.cpp

```
#include "dostep_do_io.h"
#include "dane.h"
#include "io.h"
#include "random.h"
#include "zakonczenie_obslugi_io.h"

DostepDoIO::DostepDoIO(IO** io, ZakonczenieObslugiIO* zakonczenie_obslugi_io)
 : io_(io),
   zakonczenie_obslugi_io_(zakonczenie_obslugi_io)
{
}

void DostepDoIO::Wypisz(int i){
 char buffer[255];
 Dane::ZapiszDoPliku("Zdarzenie DostepDoIO... Wykonano!\n");
 sprintf(buffer, "Wolne urzadzenie o numerze: %d\n", i);
 Dane::ZapiszDoPliku(buffer);
 sprintf(buffer, "Przypisano proces urzadzeniu nr: %d\n", i);
 Dane::ZapiszDoPliku(buffer);
 sprintf(buffer, "Zaplanowano zdarzenie ZakonczenieObslugiIO o czasie: %f\n\n",
zakonczenie_obslugi_io_->czas_[i]);
 Dane::ZapiszDoPliku(buffer);
}

void DostepDoIO::Wykonaj(int i, int iteracje){
 io_[i]->PrzydzielKolejka();
 if (Dane::GetIloscOdpowiedzi() < iteracje)
 {
  Dane::SetCalkCzasOdpowiedz(Dane::GetCalkCzasOdpowiedz() + Dane::GetCzasSymulacji() -
io_[i]->WezProces()->get_czas_czekania_io());
  Dane::SetIloscOdpowiedzi(1 + Dane::GetIloscOdpowiedzi());
 }

 int tpo = io_[i]->WezProces()->get_tpo();
 zakonczenie_obslugi_io_->czas_[i] = Dane::GetCzasSymulacji() + tpo;
 Wypisz(i);
}
```

nowy_proces.h

```
#ifndef SYMULACJA_NOWY_PROCES_H_
#define SYMULACJA_NOWY_PROCES_H_

#include "system_komputerowy.h"

//Zdarzenie generowane przez zrodlo TPG
//Umieszcza proces na koncu kolejki K1
//Zaplanowuje nowe zdarzenie NowyProces
//w konstruktorze podajemy wskaznik na system
//w systemie znajduja sie kolejki
class NowyProces
{
public:
 NowyProces(SystemKomputerowy* sys);

 //tworzy nowy proces id odaje  go do kolejki
 void Wykonaj();
 double czas_;
private:
 SystemKomputerowy* sys_;
};

#endif
```

nowy_proces.cpp

```
#include "nowy_proces.h"
#include "random.h"
#include "dane.h"

NowyProces::NowyProces(SystemKomputerowy* sys)
 :czas_(0.0),
  sys_(sys){}

void NowyProces::Wykonaj()
{
 Proces* proces = new Proces;
 proces->set_wiek(Dane::GetCzasSymulacji());
 proces->set_czas_czekania(Dane::GetCzasSymulacji());
 sys_->DodajProces(proces);
 czas_ = Random::Wykladn(4) + Dane::GetCzasSymulacji();
 fprintf(Dane::GetDoPliku(),"Zdarzenie Nowy Proces... Wykonano! \n");
 fprintf(Dane::GetDoPliku(), "Nastepny proces zaplanowano o czasie: %f\n\n", czas_);
}
```

prosba_dostepu_io.h

```
#ifndef  SYMULACJA_PROSBA_DOSTEPU_IO_H
#define SYMULACJA_PROSBA_DOSTEPU_IO_H

class IO;
class Procesor;

//Zdarzenie generowane przez czas TPIO
//zwalnia procesor oraz dodaje jego proces do kolejki io
//gdy TPIO == 0 to nie prosimy o dostep do urzadzenia io
//w konstruktorze podajemy wskaznik do tablicy procesorw i urzadzen I/O
class ProsbaDostepuIO
{
public:
 ProsbaDostepuIO(Procesor** procesory,IO** ios);

 //Wybiera losowo urzadzenie
 //zwalnia procesor
 //umieszcza proces w kolejce urzadzenia
 //jako argument num podajemy numer procesora
 void Wykonaj(int num, int czas_konca);
 double czas_[2];
private:
 Procesor** procesory_;
 IO** ios_;
};
#endif
```

prosba_dostepu_io.cpp

```
#include  "prosba_dostepu_io.h"
#include "procesor.h"
#include "dane.h"
#include "random.h"
#include "io.h"

ProsbaDostepuIO::ProsbaDostepuIO(Procesor** procesory, IO** ios)
 : czas_{-1, -1},
   procesory_(procesory),
   ios_(ios){}

void ProsbaDostepuIO::Wykonaj(int num, int czas_konca)
{
 czas_[num] = -1;

 Proces* proces = procesory_[num]->Zwolnij();
 if (czas_konca > Dane::GetCzasSymulacji())
 {
  Dane::SetCzasPracyProcesora(num, Dane::GetCzasSymulacji() - proces-
>get_czas_dostepu_proc() + Dane::GetCzasPracyProcesora(num));
 }
 int tpo = Random::Normal(1, 10, 2);
 proces->set_tpo(tpo);
 proces->set_priorytet(-tpo);

 //by wyliczyc czas pobytu w kolejce: czas_symulacji(w momencie opuszczenia kolejki) - TO
 proces->set_czas_czekania_io(Dane::GetCzasSymulacji());
 int i;
 i = Random::Normal(0, 4, 1);
 ios_[i]->DodajKolejka(proces);

 fprintf(Dane::GetDoPliku(),"Zdarzenie Prosba Dostepu do IO... Wykonano! \n");
 fprintf(Dane::GetDoPliku(), "Zwolniono Procesor nr: %d\n", num);
 fprintf(Dane::GetDoPliku(), "Dodano Proces do Kolejki urzadzenia nr: %d\n\n",i);}
```

wolny_procesor.h

```
#ifndef SYMULACJA_WOLNY_PROCESOR_H_
#define SYMULACJA_WOLNY_PROCESOR_H_

class Proces;
class WykonczProces;
class ProsbaDostepuIO;
class Procesor;
class SystemKomputerowy;

//Gdy mamy wolny procesor to przydzielamy mu proces
//z jednej z kolejek(o ile nie sa puste).
//Proces przydzielany jest na czas TPK <1:50>[ms]
//Ustawiamy czas TPIO <0,TPK0>[ms] dla zdarzenia ProsbaDostepuIO
//jesli TPIO == 0, to nie inicjujemy zdarzenia ProsbaDostepuIO
//w konstruktorze przesylam wskaznik do procesorow, systemu
// oraz zdarzen prosba_dostepu_io i wykoncz_proces
class WolnyProcesor
{
public:
 WolnyProcesor(SystemKomputerowy* sys, Procesor** procesory, ProsbaDostepuIO*
prosba_dostepu_io, WykonczProces* wykoncz_proces);

 //przydziela proces do procesora
 //kolejki z ktorych bierzemy procesy sa dobrane losowo
 //jako argument i podajemy numer procesora
 void Wykonaj(int i);

 //aktualizuje dane
 //jest czescia wywolania funkcji void Wykonaj(int i)
 void Aktualizuj(Proces* proces, int i);
private:
 SystemKomputerowy* sys_;
 Procesor** procesory_;
 ProsbaDostepuIO* prosba_dostepu_io_;
 WykonczProces* wykoncz_proces_;
};
#endif
```

wolny_procesor.cpp

```
#include "wolny_procesor.h"
#include "system_komputerowy.h"
#include "dane.h"
#include "random.h"
#include "prosba_dostepu_io.h"
#include "wykoncz_proces.h"

WolnyProcesor::WolnyProcesor(SystemKomputerowy* sys, Procesor** procesory, ProsbaDostepuIO*
prosba_dostepu_io, WykonczProces* wykoncz_proces)
 : sys_(sys),
   procesory_(procesory),
   prosba_dostepu_io_(prosba_dostepu_io),
   wykoncz_proces_(wykoncz_proces)
{
}
```

```
void WolnyProcesor::Wykonaj(int i)
{
 fprintf(Dane::GetDoPliku(), "Zdarzenie WolnyProcesor... Wykonano!\n");
 fprintf(Dane::GetDoPliku(), "Wolny Procesor o numerze: %d\n",i);
 int x = 0;
 int num = 0;
 Proces* proces;

 //Od kolejki zalezy wybor algorytmu za pomoca ktorego przydziele proces
  if (!sys_->KolejkaK()[0]->Pusta() && !sys_->KolejkaK()[1]->Pusta())
  {
         num = Random::Normal(0, 1, 3);
   x = sys_->KolejkaK()[num]->Wielkosc();
   x = Random::Normal(0, x - 1, 5);
   proces = sys_->KolejkaK()[num]->WezProces(x);
   sys_->KolejkaK()[num]->UsunProces(x);
   procesory_[i]->Przydziel(proces);
  }
  else if(!sys_->KolejkaK()[0]->Pusta())
  {
   x = sys_->KolejkaK()[0]->Wielkosc();
   x = Random::Normal(0, x - 1,5);
   proces = sys_->KolejkaK()[0]->WezProces(x);
   sys_->KolejkaK()[0]->UsunProces(x);
   procesory_[i]->Przydziel(proces);
  }
  else
  {
   proces = sys_->KolejkaK()[1]->WezProces(0);
   sys_->KolejkaK()[1]->UsunProces(0);
   procesory_[i]->Przydziel(proces);
  }
  Aktualizuj(proces,i);
}

void WolnyProcesor::Aktualizuj(Proces* proces, int i)
{
 fprintf(Dane::GetDoPliku(), "Przydzielono proces do procesora nr: %d\n", i);
 int tpw = proces->get_tpw();
 if (tpw == 0)
 {
  tpw = Random::Normal(1, 50, 6);
  proces->set_tpw(tpw);
 }
 int tpio = Random::Normal(0, tpw - 1, 6);
 if (tpio != 0)
 {
  proces->set_tpw(tpw - tpio);
  prosba_dostepu_io_->czas_[i] = Dane::GetCzasSymulacji() + tpio;
  fprintf(Dane::GetDoPliku(), "Zaplanowano zdarzenie ProsbaDostepuIO o czasie: %f \n\n",
prosba_dostepu_io_->czas_[i]);
 }
 else
 {
  wykoncz_proces_->czas_[i] = Dane::GetCzasSymulacji() + tpw;
  fprintf(Dane::GetDoPliku(), "Zaplanowano zdarzenie WykonczProces o czasie: %f\n\n",
wykoncz_proces_->czas_[i]);
 }
 proces->set_czas_czekania(Dane::GetCzasSymulacji() - proces->get_czas_czekania());
 proces->set_czas_dostepu_proc(Dane::GetCzasSymulacji());
}
```

wykoncz_proces.h

```
#ifndef SYMULACJA_WYKONCZ_PROCES_H_
#define SYMULACJA_WYKONCZ_PROCES_H_

class Procesor;

//Zdarzenie Oznacza bezpowrotne usuniecie procesu
//Wykonuje sie gdy czas TPW procesu wynosi 0
class WykonczProces
{
public:
 WykonczProces(Procesor** p);

 //bezpowrotnie zwolni proces z systemu
 //jako argument i podajemy numer procesora
 void Wykonaj(int i, int iteracje, double czas_konca);
 double czas_[2];
private:
 Procesor** p_;
};
#endif
```

wykoncz_proces.cpp

```
#include "wykoncz_proces.h"
#include "proces.h"
#include "procesor.h"
#include "dane.h"

WykonczProces::WykonczProces(Procesor** procesor_)
 : czas_{-1, -1},
   p_(procesor_)
{
}

void WykonczProces::Wykonaj(int i, int iteracje, double czas_konca)
{
 czas_[i] = -1;
 Proces* proces = p_[i]->Zwolnij();
 if (czas_konca > Dane::GetCzasSymulacji())
 {
  Dane::SetCzasPracyProcesora(i, Dane::GetCzasSymulacji() - proces->get_czas_dostepu_proc()
+ Dane::GetCzasPracyProcesora(i));
  Dane::SetCalkCzasPrzetwarzania(Dane::GetCzasSymulacji() - proces->get_wiek() +
Dane::GetCalkCzasPrzetwarzania());
 }

 if (Dane::GetCalkLiczbaProcesow() < iteracje)
 {
  Dane::SetCalkLiczbaProcesow(Dane::GetCalkLiczbaProcesow() + 1);
  if (Dane::GetMaxCzasOczek() < proces->get_czas_czekania())
   Dane::SetMaxCzasOczek(proces->get_czas_czekania());
  Dane::SetCalkCzasOczek(proces->get_czas_czekania() + Dane::GetCalkCzasOczek());
 }

 if (Dane::GetStacjonarnosc() < Dane::GetCalkLiczbaProcesow() &&
Dane::GetCalkLiczbaProcesow() < iteracje)
  fprintf(Dane::GetStats(), "%f ", proces->get_czas_czekania());

 delete proces;
 fprintf(Dane::GetDoPliku(),"Zdarzenie WykonczProces... Wykonano! \n");
 fprintf(Dane::GetDoPliku(), "Zwolniono Procesor nr: %d\n",i);
 fprintf(Dane::GetDoPliku(), "Permanentnie usunieto Proces z systemu\n\n");
}
```

zakonczenie_obslugi_io.h

```
#ifndef SYMULACJA_ZAKONCZENIE_OBSLUGI_IO_H_
#define SYMULACJA_ZAKONCZENIE_OBSLUGI_IO_H_

class SystemKomputerowy;
class IO;

//Pobudzone przez czas TPO
//Zdarzenie oznacza zakonczenie obslugi urzadzenia I/O
//Zwalnia urzadzenie, a proces kieruje do kolejki SJF
class ZakonczenieObslugiIO
{
public:
 ZakonczenieObslugiIO(IO** io, SystemKomputerowy* sys);

 //zwalnia proces z urzadzenia
 //i umieszcza go w kolejsce SJF
 //argument i oznacza numer urzadzenia
 void Wykonaj(int i);
 double czas_[5];
private:
 IO** io_;
 SystemKomputerowy* sys_;
};
#endif
```

zakonczenie_obslugi.cpp

```
#include "zakonczenie_obslugi_io.h"
#include "io.h"
#include "dane.h"
#include "system_komputerowy.h"

ZakonczenieObslugiIO::ZakonczenieObslugiIO(IO** io, SystemKomputerowy* sys)
 : czas_{-1, -1, -1, -1, -1},
   io_(io),
   sys_(sys)
{
}

void ZakonczenieObslugiIO::Wykonaj(int i)
{
   czas_[i] = -1;
   io_[i]->WezProces()->set_czas_czekania(Dane::GetCzasSymulacji() - io_[i]->WezProces()-
>get_czas_czekania());
   sys_->KolejkaK()[1]->DodajProces(io_[i]->WezProces());
   io_[i]->UsunProces();
   fprintf(Dane::GetDoPliku(), "Zdarzenie ZakonczenieObslugiIO... Wykonano! \n");
   fprintf(Dane::GetDoPliku(), "Zwolniono urzadzenie I/O nr: %d\n\n",i);

}
```

dane.h

```
#ifndef SYMULACJA_DANE_H_
#define  SYMULACJA_DANE_H_
#include <fstream>
#include <string>
class Procesor;
class IO;
class SystemKomputerowy;

//Klasa sluzaca do zbierania wynikow i wypisywania niektorych komunikator
//klasa wykonuje niektore obliczenia by ulatwic prezentacje wynikow
class Dane
{
public:
 //wypisuje w konsoli stan systemu krok po kroku
 static void GUI(Procesor** procesory, IO** io, SystemKomputerowy* sys);

 //generuje nowa nazwe pliku do ktorego zapisujemy wyniki
 static std::string ZmienNazwe();

 //wypisuje w konsoli i do pliku "Parametry.txt"
 //obliczenia symulacyjne
 static void Parametry(bool gui);
 static void Reset();

 //funkcje do zapisu danych dla roznych plikow
 static void ZapiszDoPliku(char buffer[]);

 //zeruje statystyki i rozpoczoczyna ich zbior
 static void Ustawienia();

 //funkcje get||set by zachowac enkapsulacje
 static void SetStacjonarnosc(int stat);
 static void SetCzasPomiarow(double czas);
 static void SetCzasSymulacji(double czas);
 //SetCzasPracyProcesora(numer procesora, czas);
 static void SetCzasPracyProcesora(int i,double czas);
 static void SetCalkLiczbaProcesow(int i);
 static void SetCalkCzasPrzetwarzania(double czas);
 static void SetCalkCzasOczek(double czas);
 static void SetMaxCzasOczek(double czas);
 static void SetKernel(int x);
 static void SetL(double L);
 static void SetCalkCzasOdpowiedz(double czas);
 static void SetIloscOdpowiedzi(int i);
 static void SetNumerSymulacji(int n);

 static void SetDoPliku(FILE* plik);
 static void SetStats(FILE* plik);

 static int GetStacjonarnosc();
 static double GetCzasPomiarow();
 static double GetCzasSymulacji();
 //GetCzasPracyProcesora(numer procesora);
 static double GetCzasPracyProcesora(int i);
 static int GetCalkLiczbaProcesow();
 static double GetCalkCzasPrzetwarzania();
 static double GetCalkCzasOczek();
 static double GetMaxCzasOczek();
 static double GetCalkCzasOdpowiedz();
 static int GetIloscOdpowiedzi();
 static int GetNumerSymulacji();
```

```
 static FILE* GetStats();
 static FILE* GetDoPliku();

 static void SetCzasIteracje(double czas, int iteracje);
private:
 static int stacjonarnosc_;

 static double przepustowosc_;
 static double przetwarzanie_;

 //czas trwania pomiaru
 static double czas_pomiarow_;

 //calkowity czas symulacji
 static double czas_symulacji_;

 static bool wyzeruj_wyniki_;

 //czas jaki procesor jest uzywany przez procesy
 static double czas_pracy_procesora_[2];

 //liczba mowiaca ile stworzylismy procesow
 static int calk_liczba_procesow_;

 //suma czasu jaki uplywa miedzy zgloszeniem procesu do systemu,
 //a jego zakonczeniem (czas zycia wszystkich procesow)
 static double calk_czas_przetwarzania_;

 //suma czasu oczekiwania na procesor przez procesy oraz counter
 static double calk_czas_oczek_na_procesor_;
 static double max_czas_oczek_;

 //parametry generatora
 static int kernel_;
 static double L_;

 static int liczba_iteracji_;
 static double czas_konca_;

 //czas miedzy zgloszeniem zadania dostepu do jednego z urzadzen io
 //i jego otrzymaniem (czas spedzony w kolejkach io przez wszystkie procesy)
 //oraz counter
 static double calk_czas_odpowiedzi_;
 static int ilosc_odpowiedzi_;

 static int numer_symulacji_;

 //plik do zapisywania danych - step by step
 static FILE* do_pliku_;

 //plik do koncowych statystyk
 static FILE* stats_;
};
#endif
```

dane.cpp

```
#include  "dane.h"
#include "system_komputerowy.h"
#include "fstream"
#include "stdio.h"
```

```
FILE* Dane::do_pliku_ = nullptr;
FILE* Dane::stats_ = nullptr;
double Dane::czas_pomiarow_ = 0.0;
double Dane::czas_symulacji_ = 0.0;
double Dane::czas_pracy_procesora_[] = { 0.0, 0.0 };
double Dane::calk_czas_przetwarzania_ = 0.0;
double Dane::calk_czas_oczek_na_procesor_ = 0.0;
double Dane::max_czas_oczek_ = 0.0;
double Dane::calk_czas_odpowiedzi_ = 0.0;
double Dane::L_ = 0.0;
int Dane::calk_liczba_procesow_ = 0;
int Dane::ilosc_odpowiedzi_ = 0;
int Dane::kernel_ = 0;
int Dane::numer_symulacji_ = 0;
int Dane::stacjonarnosc_ = 0;
bool Dane::wyzeruj_wyniki_ = true;
int Dane::liczba_iteracji_ = 0;
double Dane::czas_konca_ = 0;
double Dane::przetwarzanie_ = 0;
double Dane::przepustowosc_ = 0;

void Dane::GUI(Procesor** procesory, IO** io, SystemKomputerowy* sys)
{
 printf("\n\nProcesory:\n   ");
 for (int i = 0; i < 2; i++)
  if (procesory[i]->Wolny())
   printf(" [ ]");
  else
   printf(" [X]");
 printf("\n\nUrzadzenia wejscia wyjscia:  ");
 for (int i = 0; i < 5; i++)
 {
  printf("\n   io[%d]", i + 1);
  if (io[i]->Wolny())
   printf(" [ ]   kolejka urzadzenia: <->");
  else
  {
   printf(" [X]   kolejka urzadzenia: <-");
   for (int j = 0; j < io[i]->WielkoscKolejki(); j++)
    printf("x-");
   printf(">");
  }
 }
 printf("\n\nKolejka SJF:  ");
 printf("<-");
 for (int j = 0; j < sys->KolejkaK()[1]->Wielkosc(); j++)
  printf("x-");
 printf(">");
 printf("\n\nKolejka dla Nowych:  ");
 printf("<-");
 for (int j = 0; j < sys->KolejkaK()[0]->Wielkosc(); j++)
  printf("x-");
 printf(">\n\n\n");

 std::system("Pause");
}

std::string Dane::ZmienNazwe()
{
 std::string plik = "symulacja_";
 std::string nr_sym = std::to_string(Dane::numer_symulacji_);
  std::string nazwa = plik + nr_sym + ".txt";
  return nazwa;
}
```

```
void Dane::Parametry(bool gui)
{
 if (gui)
 {
  printf("Maksymalny czas oczekiwania na procesor: %f\n",max_czas_oczek_);
  printf("Sredni czas oczekiwania na procesor: %f\n", (calk_liczba_procesow_) ?
calk_czas_oczek_na_procesor_ / calk_liczba_procesow_ : 0);
  for (int i = 0; i < 2; i++)
   printf("Wykorzystanie procesora nr %d wynosi: %f %%\n", i, czas_symulacji_ ?
czas_pracy_procesora_[i] / czas_symulacji_ * 100 : 0);
  printf("Przepustowosc systemu mierzona liczba procesow zakonczonych w jednostce czasu:
%f\n",czas_symulacji_ ? calk_liczba_procesow_ / czas_symulacji_ : 0);
  printf("Sredni czas przetwarzania(czas miedzy zgloszeniem procesu do systemu a jego
zakonczeniem): %f\n",(calk_liczba_procesow_) ? calk_czas_przetwarzania_ /
calk_liczba_procesow_ : 0);
  printf("Sredni czas odpowiedzi(czas miedzy zgloszeniem zadania dostepu do IO, a jego
otrzymaniem): %f\n", ilosc_odpowiedzi_ ? calk_czas_odpowiedzi_ / ilosc_odpowiedzi_ : 0);
  printf("\n-------------------------------------------------------------------------\n");
 }
  fprintf(do_pliku_,
"\n-------------------------------------------------------------------------\n");
  fprintf(do_pliku_, "Maksymalny czas oczekiwania na procesor: %f\n", max_czas_oczek_);
  fprintf(do_pliku_,"Sredni czas oczekiwania na procesor: %f\n", (calk_liczba_procesow_) ?
calk_czas_oczek_na_procesor_ / calk_liczba_procesow_ : 0);
  for (int i = 0; i < 2; i++)
   fprintf(do_pliku_,"Wykorzystanie procesora nr %d wynosi: %f %%\n", i, czas_pomiarow_ ?
czas_pracy_procesora_[i] / czas_pomiarow_ * 100 : 0);
  fprintf(do_pliku_,"Przepustowosc systemu mierzona liczba procesow zakonczonych w jednostce
czasu: %f\n", czas_pomiarow_ ? calk_liczba_procesow_ / czas_pomiarow_ : 0);
  fprintf(do_pliku_,"Sredni czas przetwarzania(czas miedzy zgloszeniem procesu do systemu a
jego zakonczeniem): %f\n", (calk_liczba_procesow_) ? calk_czas_przetwarzania_ /
calk_liczba_procesow_ : 0);
  fprintf(do_pliku_,"Sredni czas odpowiedzi(czas miedzy zgloszeniem zadania dostepu do IO, a
jego otrzymaniem): %f\n", ilosc_odpowiedzi_ ? calk_czas_odpowiedzi_ / ilosc_odpowiedzi_ :
0);
  fprintf(do_pliku_, "czas konca: %f, czas pracy 1: %f\n", czas_konca_,
czas_pracy_procesora_[0]);
}



void Dane::Reset()
{

 fprintf(stats_, "\nMaksymalny czas oczekiwania na procesor: %f\n", max_czas_oczek_);
 fprintf(stats_, "Sredni czas oczekiwania na procesor: %f\n", (liczba_iteracji_) ?
calk_czas_oczek_na_procesor_ / liczba_iteracji_ : 0);
 for (int i = 0; i < 2; i++)
  fprintf(stats_, "Wykorzystanie procesora nr %d wynosi: %f %%\n", i, czas_konca_ ?
czas_pracy_procesora_[i] / czas_konca_ * 100 : 0);
 fprintf(stats_, "Przepustowosc systemu mierzona liczba procesow zakonczonych w jednostce
czasu: %f\n", przepustowosc_);
 fprintf(stats_, "Sredni czas przetwarzania(czas miedzy zgloszeniem procesu do systemu a
jego zakonczeniem): %f\n", przetwarzanie_);
 fprintf(stats_, "Sredni czas odpowiedzi(czas miedzy zgloszeniem zadania dostepu do IO, a
jego otrzymaniem): %f\n", ilosc_odpowiedzi_ ? calk_czas_odpowiedzi_ / ilosc_odpowiedzi_ :
0);
 fprintf(stats_, "\n-------------------------------------\n");
 fclose(stats_);
 max_czas_oczek_ = 0.0;
 czas_pomiarow_ = 0.0;
 czas_symulacji_ = 0.0;
 czas_pracy_procesora_[0] = 0.0;
```

```
 czas_pracy_procesora_[1] = 0.0;
 calk_czas_przetwarzania_ = 0.0;
 calk_czas_oczek_na_procesor_ = 0.0;
 calk_czas_odpowiedzi_ = 0.0;
 L_ = 0.0;
 calk_liczba_procesow_ = 0;
 ilosc_odpowiedzi_ = 0;
 kernel_ = 0;
 numer_symulacji_++;

 stacjonarnosc_ = 0;
 wyzeruj_wyniki_ = true;
 przetwarzanie_ = 0;
 przepustowosc_ = 0;
}

void Dane::ZapiszDoPliku(char buffer[])
{
 fprintf(do_pliku_, buffer);
}

void Dane::Ustawienia()
{
 if (calk_liczba_procesow_ >= stacjonarnosc_ && wyzeruj_wyniki_)
 {
  wyzeruj_wyniki_ = false;
  czas_pomiarow_ = 0.0;
  max_czas_oczek_ = 0.0;
  czas_pracy_procesora_[0] = 0.0;
  czas_pracy_procesora_[1] = 0.0;
  calk_czas_przetwarzania_ = 0.0;
  calk_czas_oczek_na_procesor_ = 0.0;
  calk_czas_odpowiedzi_ = 0.0;
  calk_liczba_procesow_ = 0;
  przepustowosc_ = 0;
  przetwarzanie_ = 0;
  ilosc_odpowiedzi_ = 0;

 }
}

void Dane::SetStacjonarnosc(int stat)
{
 stacjonarnosc_ = stat;
}

void Dane::SetCzasPomiarow(double czas)
{
 czas_pomiarow_ = czas;
}

void Dane::SetCzasSymulacji(double czas)
{
 czas_symulacji_ = czas;
}

void Dane::SetCzasPracyProcesora(int i, double czas)
{
 czas_konca_ = czas_symulacji_;
 czas_pracy_procesora_[i] = czas;
}

void Dane::SetCalkLiczbaProcesow(int i)
{
 calk_liczba_procesow_ = i;
```

```
}

void Dane::SetCalkCzasPrzetwarzania(double czas)
{
 calk_czas_przetwarzania_ = czas;
 przetwarzanie_ = calk_czas_przetwarzania_ /calk_liczba_procesow_;
}

void Dane::SetCalkCzasOczek(double czas)
{
 przepustowosc_ = static_cast<double>(liczba_iteracji_) / czas_pomiarow_;
 calk_czas_oczek_na_procesor_ = czas;
}


void Dane::SetMaxCzasOczek(double czas)
{
 max_czas_oczek_ = czas;
}

void Dane::SetKernel(int x)
{
 kernel_ = x;
}

void Dane::SetL(double L)
{
 L_ = L;
}

void Dane::SetCalkCzasOdpowiedz(double czas)
{
 calk_czas_odpowiedzi_ = czas;
}

void Dane::SetIloscOdpowiedzi(int i)
{
 ilosc_odpowiedzi_ = i;
}

void Dane::SetNumerSymulacji(int n)
{
 numer_symulacji_ = n;
}

void Dane::SetDoPliku(FILE* plik)
{
 do_pliku_ = plik;
}

void Dane::SetStats(FILE* plik)
{
 stats_ = plik;
}

int Dane::GetStacjonarnosc()
{
 return stacjonarnosc_;
}

double Dane::GetCzasPomiarow()
{
 return czas_pomiarow_;
}
```

```
double Dane::GetCzasSymulacji()
{
 return czas_symulacji_;
}

double Dane::GetCzasPracyProcesora(int i)
{
 return czas_pracy_procesora_[i];
}

int Dane::GetCalkLiczbaProcesow()
{
 return calk_liczba_procesow_;
}

double Dane::GetCalkCzasPrzetwarzania()
{
 return calk_czas_przetwarzania_;
}

double Dane::GetCalkCzasOczek()
{
 return calk_czas_oczek_na_procesor_;
}


double Dane::GetMaxCzasOczek()
{
 return max_czas_oczek_;
}


double Dane::GetCalkCzasOdpowiedz()
{
 return calk_czas_odpowiedzi_;
}

int Dane::GetIloscOdpowiedzi()
{
 return ilosc_odpowiedzi_;
}

int Dane::GetNumerSymulacji()
{
 return numer_symulacji_;
}

FILE* Dane::GetStats()
{
 return stats_;
}

FILE* Dane::GetDoPliku()
{
 return do_pliku_;
}

void Dane::SetCzasIteracje(double czas, int iteracje)
{
 liczba_iteracji_ = iteracje;
 czas_konca_ = czas;
}
```

io.h

```
#ifndef SYMULACJA_IO_H_
#define SYMULACJA_IO_H_
#include "kolejka.h"

//reprezentuje urzadzenia I/O ktore sa zajmowane przez procesy
//uzycie: IO io();
//io.DodajKolejka(p);
//io.UaktualnijPriorytet();
//io.PrzydzielKolejka();
//io.UsunProces();
class IO
{
public:
 IO();
 ~IO();
 //dodaje proces do kolejki priorytetowej
 void DodajKolejka(Proces* proces);

 //przydziela proces z kolejki urzadzenia
 void PrzydzielKolejka();

 //zwalnia urzadzenie
 void UsunProces();

 //uaktualnia priorytety procesow  w kolejkach do urzadzen
 void UaktualnijPriorytet();

 //zwraca proces obslugujacy I/O
 Proces* WezProces();
 int WielkoscKolejki();
 bool Wolny();
private:
 Proces* obecny_proces_;
 KolejkaPrio* kolejka_priorytetowa_;
};

#endif
```

io.cpp

```
#include "io.h"
#include "dane.h"

IO::IO()
 : obecny_proces_(nullptr),
   kolejka_priorytetowa_(new KolejkaPrio()){}

IO::~IO()
{
 if (obecny_proces_)
  delete obecny_proces_;
 delete kolejka_priorytetowa_;
}

void IO::DodajKolejka(Proces* proces)
{
 kolejka_priorytetowa_->DodajProces(proces);
}



//przydziela proces urzadzeniu prosto z jego kolejki
void IO::PrzydzielKolejka()
{
 UaktualnijPriorytet();
 obecny_proces_ = kolejka_priorytetowa_->Wyrzuc();
}

void IO::UsunProces()
{
 obecny_proces_ = nullptr;
}

void IO::UaktualnijPriorytet()
{
 kolejka_priorytetowa_->Uaktualnij();
 KolejkaPrio* kolejka = new KolejkaPrio();
 KolejkaPrio* ptr;
 while(kolejka_priorytetowa_->Wielkosc() != 0)
  kolejka->DodajProces(kolejka_priorytetowa_->Wyrzuc());
 ptr = kolejka_priorytetowa_;
 kolejka_priorytetowa_ = kolejka;
 delete ptr;
}

Proces* IO::WezProces()
{
 return obecny_proces_;
}

int IO::WielkoscKolejki()
{
 return kolejka_priorytetowa_->Wielkosc();
}

bool IO::Wolny()
{
 if (obecny_proces_)
  return false;
 else return true;
}
```

kolejka.h

```
#ifndef SYMULACJA_KOLEJKA_H_
#define SYMULACJA_KOLEJKA_H_
#include "proces.h"
#include <queue>

//Atrybut kolejki priorytetowej
class Pole
{
public:
 Pole();
 ~Pole();
 Pole(Proces* proces);
private:
 Proces* proces_;
 Pole* nastepne_;
 friend class KolejkaPrio;
 friend class SJF;
};

//Kolejka do której trafiają procesy zwolnione
//przez procesor z powodu żądania dostępu do
//urządzeń wej - wyj.Procesy są w niej poukładane
//w zależności od ich priorytetu.
//uzycie:
//KolejkaPrio kolejka;
//kolejka.DodajProces(p);
class KolejkaPrio
{
public:
 KolejkaPrio();
 virtual ~KolejkaPrio();
 virtual void DodajProces(Proces* proces);

 //uaktualnia priorytety kolejki
 void Uaktualnij();

 //Wyrzuca proces z listy
 //proces NIE JEST trwale usuniety z systemu
 //zwraca wyrzucony proces
 Proces* Wyrzuc();
 //zwraca liczebnosc listy
 virtual int Wielkosc();
protected:
 //liczba procesow w liscie
 int i;
 Pole* lista_;
};

//Kolejka przechowuja procesy zwolnione z urzadzen I/O
//Procesy poukładane są zgodnie z metodą SJF
//-najpierw najkrótszy pozostały czas.
class SJF:public KolejkaPrio
{
public:
 SJF();
 virtual ~SJF();
 virtual void DodajProces(Proces* proces);
 virtual int Wielkosc();
 virtual Proces* WezProces(int x);
 virtual void UsunProces(int x);
 virtual bool Pusta();
```

```
 int i;
};

//kolejka do ktorej trafiaja procesy ktore dopiero
//pojawily sie w systemie i nie mialy jeszcze
//przydzielonego procesora.
//przyklad uzycia K kolejka;
class Kolejka : public SJF
{
public:
 ~Kolejka();
 void DodajProces(Proces* proces);
 int Wielkosc();
 Proces* WezProces(int x);
 void UsunProces(int x);
 bool Pusta();
private:
 std::deque<Proces*> lista_;
 int i;
};


#endif
```

kolejka.cpp

```
#include "kolejka.h"
#include "dane.h"
#include "random.h"

Kolejka::~Kolejka()
{
 Proces* proces;
 while(lista_.size() != 0)
 {
  proces = lista_.back();
  lista_.pop_back();
  delete proces;
 }
}

void Kolejka::DodajProces(Proces* proces)
{
 lista_.insert(lista_.begin(),proces);
}

int Kolejka::Wielkosc()
{
 return lista_.size();
}

Proces* Kolejka::WezProces(int x)
{
 return lista_[x];
}

void Kolejka::UsunProces(int x)
{
 lista_.erase(lista_.begin()+x);
}

bool Kolejka::Pusta()
{
 return lista_.empty();
```

```
}


Pole::Pole():proces_(nullptr), nastepne_(nullptr){}

Pole::~Pole()
{
 if(proces_)
  delete proces_;
 proces_ = nullptr;
}

Pole::Pole(Proces* proces): proces_(proces), nastepne_(nullptr) {}

KolejkaPrio::KolejkaPrio():i(0), lista_(new Pole()){}

KolejkaPrio::~KolejkaPrio()
{
 Pole* ptr = lista_;
 while(ptr)
 {
  ptr = ptr->nastepne_;
  delete lista_;
  lista_ = ptr;
 }
 lista_ = nullptr;
}

void KolejkaPrio::DodajProces(Proces* proces)
{
 Pole* ptr = lista_;
 Pole* nowy = new Pole(proces);
 while (ptr->nastepne_)
 {
  if (proces->get_priorytet() > lista_->nastepne_->proces_->get_priorytet())
   break;
  ptr = ptr->nastepne_;
 }
 nowy->nastepne_ = ptr->nastepne_;
 ptr->nastepne_ = nowy;
 ++i;
}

SJF::SJF() :KolejkaPrio(){}

void SJF::DodajProces(Proces* proces)
{
 Pole* ptr = lista_;
 Pole* nowy = new Pole(proces);
 while (ptr->nastepne_)
 {
  if (proces->get_tpw() < lista_->nastepne_->proces_->get_tpw())
   if (abs(proces->get_tpw() - lista_->nastepne_->proces_->get_tpw()) < 0.000001 )
   {
    int x = Random::Normal(0, 1,0);
    if (x)
     break;
   }
   else
    break;
  ptr = ptr->nastepne_;
 }
 nowy->nastepne_ = ptr->nastepne_;
 ptr->nastepne_ = nowy;
 ++i;
```

```
}

SJF::~SJF()
{
 Pole* ptr = lista_;
 while (ptr)
 {
  ptr = ptr->nastepne_;
  delete lista_;
  lista_ = ptr;
 }
 lista_ = nullptr;
}


int SJF::Wielkosc()
{
 return i;
}

Proces* SJF::WezProces(int x)
{
 return lista_->nastepne_->proces_;
}

void SJF::UsunProces(int x)
{
 lista_->nastepne_ = lista_->nastepne_->nastepne_;
 --i;
}

bool SJF::Pusta()
{
 if (lista_->nastepne_ == nullptr)return true;
 else return false;
}


void KolejkaPrio::Uaktualnij()
{
 Pole* ptr = lista_;
 double tpo;
 double to;
 while(ptr->nastepne_)
 {
  tpo = ptr->nastepne_->proces_->get_tpo();
  to = Dane::GetCzasSymulacji() - ptr->nastepne_->proces_->get_czas_czekania_io();
  ptr->nastepne_->proces_->set_priorytet(to - tpo);
  ptr = ptr->nastepne_;
 }
}

//usuwa proces z kolejki
//przydziela usuniety proces urzadzeniu
Proces* KolejkaPrio::Wyrzuc()
{
 Proces* ptr = lista_->nastepne_->proces_;
 lista_->nastepne_ = lista_->nastepne_->nastepne_;
 --i;
 return ptr;
}

int KolejkaPrio::Wielkosc()
{
 return i;
```

```
}
```

model.h

```
#ifndef SYMULACJA_MODEL_H_
#define SYMULACJA_MODEL_H_
#include "io.h"
#include "system_komputerowy.h"
#include "dostep_do_io.h"
#include "wolny_procesor.h"
#include "wykoncz_proces.h"
#include "zakonczenie_obslugi_io.h"
#include "nowy_proces.h"
#include "prosba_dostepu_io.h"
#include <fstream>


//glowna petla symulacyjna i obsluga zdarzen
class Model
{
public:
 //jako argument Modelu podajemy liczbe iteracji
 //gdy model wykona okreslona liczbe iteracji, konczy prace
 Model();
 ~Model();

 //rozpoczynamy symulacje
 void Wykonaj();
 void Menu();

 bool ZdarzenieNowyProces();
 bool ZdarzenieProsbaDostepuIO();
 bool ZdarzenieZakonczenieObslugiIO();
 bool ZdarzenieWykonczProces();
 bool ZdarzenieWolnyProcesor();
 bool ZdarzenieGotoweUrzadzenie();

 //decyduje ot rybie graficznym
 //decyduje o momencie rozpoczecia liczenia statystyk
 void Ustawienia(int ite);
 bool Powtorzyc();
 bool Koniec(int ite);

 //aktualizujemy czasy i priorytety
 //w razie niepowodzenia zwracamy false i konczymy symulacje
 void Aktualizuj();
private:
 double czas_;
 int iteracje_;
 double czas_konca_;

 //liczba procesorow
 const int k_p_;

 //liczba io
 const int k_io_;

 const int k_epsilon_;

 //gui oznacza tryb krokowy
 int gui_;

 IO* io_[5];
 SystemKomputerowy* moj_system_;
 Procesor* p_[2];
```

```
 NowyProces* nowy_proces_event_;
 ProsbaDostepuIO* prosba_dostepu_io_event_;
 ZakonczenieObslugiIO* zakonczenie_obslugi_io_event_;
 WykonczProces* wykoncz_proces_event_;
 WolnyProcesor* wolny_procesor_event_;
 DostepDoIO* dostep_do_io_;
};
#endif
```

model.cpp

```
#define __STDC_FORMAT_MACROS
#include "model.h"
#include "random.h"
#include "dane.h"
#include <iostream>
#include <stdio.h>
#include <fstream>
#include <inttypes.h>

Model::Model()
 : czas_(0.0),
   iteracje_(60000),
   czas_konca_(200000),
   k_p_(2),
   k_io_(5),
   k_epsilon_(0.00001),
   gui_(0)
{
 for (int i = 0; i < k_io_; i++)
  io_[i] = new IO();
 moj_system_ = new SystemKomputerowy();
 p_[0] = new Procesor();
 p_[1] = new Procesor();

 nowy_proces_event_ = new NowyProces(moj_system_);
 prosba_dostepu_io_event_ = new ProsbaDostepuIO(p_, io_);
 zakonczenie_obslugi_io_event_ = new ZakonczenieObslugiIO(io_, moj_system_);
 wykoncz_proces_event_ = new WykonczProces(p_);
 wolny_procesor_event_ = new WolnyProcesor(moj_system_, p_, prosba_dostepu_io_event_,
wykoncz_proces_event_);
 dostep_do_io_ = new DostepDoIO(io_, zakonczenie_obslugi_io_event_);
}
Model::~Model()
{

 for (int i = 0; i < k_io_; i++)
  delete io_[i];

 delete moj_system_;
 delete p_[0];
 delete p_[1];

 delete nowy_proces_event_;
 delete prosba_dostepu_io_event_;
 delete zakonczenie_obslugi_io_event_;
 delete wykoncz_proces_event_;
 delete wolny_procesor_event_;
 delete dostep_do_io_;
}


void Model::Wykonaj()
{
```

```
 int ite = 0;
 bool flaga = true;
 for (;;)
 {
  fprintf(Dane::GetDoPliku(),"\n\n      ---Czas Systemu: %f ---\n",
Dane::GetCzasSymulacji());
  flaga = true;
  while (flaga)
  {
   flaga = false;

   if (ZdarzenieNowyProces())
    flaga = true;

   if (ZdarzenieProsbaDostepuIO())
    flaga = true;

   if (ZdarzenieZakonczenieObslugiIO())
    flaga = true;

   if (ZdarzenieWykonczProces())
    flaga = true;

   if (ZdarzenieWolnyProcesor())
    flaga = true;

   if (ZdarzenieGotoweUrzadzenie())
    flaga = true;
  }

  //tu zmienilem w petli uwaga
  Ustawienia(ite);

  //tu tez zmienialem, dobrze sprwadzic
  if (Koniec(ite))
   break;
  Aktualizuj();
  ite++;
 }
}

void Model::Menu()
{
 int64_t kernel = Random::KernelNext();
 double L = 0.073;
 int stacjonarnosc;
 if(Dane::GetNumerSymulacji() == 0)
 {
  std::cout << "\n\nPodaj kernel (np. 1129): ";
  std::cin >> kernel;
  fprintf(Dane::GetDoPliku(), "Kernel: %" PRId64 "\n", kernel);
  std::cout << "Podaj intensywnosc L (np. 0.073): ";
  std::cin >> L;
  fprintf(Dane::GetDoPliku(), "L: %f\n", L);
  std::cout << "Podaj ilosc iteracji dla procesow (np. 50 000): ";
  std::cin >> iteracje_;
  std::cout << "Podaj czas symulacji (np. 200 000): ";
  std::cin >> czas_konca_;

  Dane::SetCzasIteracje(czas_konca_, iteracje_);
 }
 else {
  std::cout << "\n\nPodaj kernel (sugerowany " <<kernel<<" ): ";
  std::cin >> kernel;
  fprintf(Dane::GetDoPliku(), "Kernel: %" PRId64 "\n", kernel);
```

```
  std::cout << "Podaj intensywnosc L (np. 0.073): ";
  std::cin >> L;
  fprintf(Dane::GetDoPliku(), "L: %f\n", L);
  std::cout << "Podaj ilosc iteracji dla procesow (np. 50 000): ";
  std::cin >> iteracje_;
  std::cout << "Podaj czas symulacji (np. 200 000): ";
  std::cin >> czas_konca_;

  Dane::SetCzasIteracje(czas_konca_, iteracje_);
 }
 fprintf(Dane::GetDoPliku(), "Liczba iteracji: %d\n",iteracje_);
 std::cout << "\nLiczba zakonczonych procesow wymagana do rozpoczecia rejetrowania
wynikow:\n(0 - od poczatku)\nWybierasz: ";
 std::cin >> stacjonarnosc; Dane::SetStacjonarnosc(stacjonarnosc);


 std::cout << "\nSymulacja Natychmiastowa: Wprowadz '0'\n";
 std::cout << "Symulacja Krok po kroku: Wprowadz '1'\n";
 std::cout << "Wybierasz:  ";
 std::cin >> gui_;
 Random::Init(kernel, L);
}

bool Model::ZdarzenieNowyProces()
{
 if (abs(nowy_proces_event_->czas_ - Dane::GetCzasSymulacji()) <= k_epsilon_)
 {
  nowy_proces_event_->Wykonaj();
  return true;
 }
 return false;
}

bool Model::ZdarzenieProsbaDostepuIO()
{
 bool wykonano = false;
 for (int i = 0; i < k_p_; i++)
  if (abs(prosba_dostepu_io_event_->czas_[i] - Dane::GetCzasSymulacji()) <= k_epsilon_)
  {
   prosba_dostepu_io_event_->Wykonaj(i, czas_konca_);
   wykonano =  true;
  }
 return wykonano;
}

bool Model::ZdarzenieZakonczenieObslugiIO()
{
 bool wykonano = false;
 for (int i = 0; i < k_io_; i++)
  if (abs(zakonczenie_obslugi_io_event_->czas_[i] - Dane::GetCzasSymulacji()) <= k_epsilon_)
  {
   zakonczenie_obslugi_io_event_->Wykonaj(i);
   wykonano = true;
  }
 return wykonano;
}

bool Model::ZdarzenieWykonczProces()
{
 bool wykonano = false;
 for (int i = 0; i < k_p_; i++)
  if (abs(wykoncz_proces_event_->czas_[i] - Dane::GetCzasSymulacji()) <= k_epsilon_)
  {
   wykoncz_proces_event_->Wykonaj(i, iteracje_, czas_konca_);
   wykonano = true;
```

```
  }
 return wykonano;
}

bool Model::ZdarzenieWolnyProcesor()
{
 bool wykonano = false;
 for (int i = 0; i < k_p_; i++)
  if ((!moj_system_->KolejkaK()[0]->Pusta() || !moj_system_->KolejkaK()[1]->Pusta()) &&
p_[i]->Wolny())
  {
   wolny_procesor_event_->Wykonaj(i);
   wykonano = true;
  }
 return wykonano;
}

bool Model::ZdarzenieGotoweUrzadzenie()
{
 bool wykonano = false;
 for (int i = 0; i < k_io_; i++)
 {
  if (io_[i]->Wolny() && io_[i]->WielkoscKolejki())
  {
   dostep_do_io_->Wykonaj(i, iteracje_);
   wykonano = true;
  }
 }
 return wykonano;
}

void Model::Ustawienia(int ite)
{
 if (gui_)
  Dane::GUI(p_, io_, moj_system_);
 Dane::Parametry(gui_);

 //ustawienie rejestrowania wynikow
 Dane::Ustawienia();

}

bool Model::Powtorzyc()
{
 bool restart = 1;
 std::cout << "\nPonowic Symulacje: '1'\n";
 std::cout << "Wyjdz: '0'\n";
 std::cout << "Wybierasz:  ";
 std::cin >> restart;
 std::cout << "\n---------------------------------------\n";
 return restart;
}

bool Model::Koniec(int ite)
{
 if (Dane::GetCalkLiczbaProcesow() >= iteracje_ && Dane::GetCzasSymulacji() >= czas_konca_)
{
  fprintf(Dane::GetDoPliku(), "            ---Koniec---\n");
  return true;
 }
 return false;
}
```

```
void Model::Aktualizuj()
{
  double tmin = nowy_proces_event_->czas_;
 for (int i = 0; i < k_p_; i++)
  if (prosba_dostepu_io_event_->czas_[i] >= 0 && prosba_dostepu_io_event_->czas_[i] < tmin)
   tmin = prosba_dostepu_io_event_->czas_[i];
 for (int i = 0; i < k_io_; i++)
  if (zakonczenie_obslugi_io_event_->czas_[i] >= 0 && zakonczenie_obslugi_io_event_-
>czas_[i] < tmin)
   tmin = zakonczenie_obslugi_io_event_->czas_[i];
 for (int i = 0; i < k_p_; i++)
  if (wykoncz_proces_event_->czas_[i] >= 0 && wykoncz_proces_event_->czas_[i] < tmin)
   tmin = wykoncz_proces_event_->czas_[i];

 tmin = tmin - Dane::GetCzasSymulacji();
 Dane::SetCzasSymulacji(Dane::GetCzasSymulacji() + tmin);
 Dane::SetCzasPomiarow(Dane::GetCzasPomiarow() + tmin);
}
```

proces.h

```
#ifndef SYMULACJA_PROCES_H_
#define SYMULACJA_PROCES_H_

//reprezentuje procesy ktore pojawiaja sie w losowych chwilach
//przyklad stworzenia procesu: Proces p();
class Proces
{
public:
 Proces();
 //funkcje sluza do zwracania i przypisywania zmiennych w obiekcie
 int get_tpw();
 int get_tpo();
 int get_priorytet();
 double get_wiek();
 double get_czas_zgloszen();
 double get_czas_czekania();
 double get_czas_czekania_io();
 double get_czas_dostepu_proc();
 void set_tpo(int x);
 void set_tpw(int x);
 void set_priorytet(double x);
 void set_wiek(double x);
 void set_czas_zgloszen(double x);
 void set_czas_czekania_io(double x);
 void set_czas_czekania(double x);
 void set_czas_dostepu_proc(double x);
private:
 //czas dostepu do procesora zadany przez proces
 int tpw_;

 //czas nieprzerwanego dostepu do I/O
 int tpo_;

 int priorytet_;

 //czas stworzenia procesu
 double wiek_;

 //czas zgloszenia zadania dostepu
 double czas_zgloszen_;

 //czas uzyskania dostepu do procesora
 double czas_dostepu_proc_;
```

```
 //czas czekania w kolejce procesora
 double czas_czekania_;

 //czas czekania w kolejce io
 double czas_czekania_io_;
};
#endif//SYMULACJA_PROCES_H_
```

proces.cpp

```
#include "proces.h"
#include <cstdio>

Proces::Proces():
 tpw_(0),
 tpo_(0),
 priorytet_(0),
 wiek_(0.0),
 czas_zgloszen_(0.0),
 czas_czekania_(0.0)
{
 ////printf("[SYSTEM]: stworzono Proces\n");
}

int Proces::get_tpw()
{
 return tpw_;
}

int Proces::get_tpo()
{
 return tpo_;
}

int Proces::get_priorytet()
{
 return priorytet_;
}

double Proces::get_wiek()
{
 return wiek_;
}

double Proces::get_czas_zgloszen()
{
 return czas_zgloszen_;
}

double Proces::get_czas_czekania()
{
 return czas_czekania_;
}

double Proces::get_czas_czekania_io()
{
 return czas_czekania_io_;
}

double Proces::get_czas_dostepu_proc()
{
 return czas_dostepu_proc_;
}
```

```
void Proces::set_tpo(int x)
{
 tpo_ = x;
}

void Proces::set_tpw(int x)
{
 tpw_ = x;
}

void Proces::set_priorytet(double x)
{
 priorytet_ = x;
}

void Proces::set_wiek(double x)
{
 wiek_ = x;
}

void Proces::set_czas_czekania(double x)
{
 czas_czekania_ = x;
}

void Proces::set_czas_zgloszen(double x)
{
 czas_zgloszen_ = x;
}

void Proces::set_czas_czekania_io(double x)
{
 czas_czekania_io_ = x;
}

void Proces::set_czas_dostepu_proc(double x)
{
 czas_dostepu_proc_ = x;
}
```

procesor.h

```
#ifndef SYMULACJA_PROCESOR_H_
#define SYMULACJA_PROCESOR_H_
#include "proces.h"

//reprezentuje pojedynczy procesor ktory jest zajmowany przez proces
//uzycie Procesor proc();
//Procesor proc;
//proc.Przydziel(p);
//proc.Zwolnij();
//funkcje nie wymagaja dodatkowego komentarza
class Procesor
{
public:
 Procesor();
 ~Procesor();
 void Przydziel(Proces* x);
 Proces* Zwolnij();
 bool Wolny();
private:
 Proces* obecny_proces_;
};
#endif
```

procesor.cpp

```
#include "procesor.h"
#include <cstdio>
#include <cstdlib>

Procesor::Procesor()
 : obecny_proces_(nullptr)
{

}

Procesor::~Procesor()
{
 if (obecny_proces_)
  delete obecny_proces_;
}


void Procesor::Przydziel(Proces* x)
{
 obecny_proces_ = x;
}

Proces* Procesor::Zwolnij()
{
 Proces* p = obecny_proces_;
  obecny_proces_ = nullptr;
  return p;
}

bool Procesor::Wolny()
{
 if (obecny_proces_ == nullptr)
  return true;
 else return false;
}
```

random.h

```
#ifndef SYMULACJA_RANDOM_H_
#define SYMULACJA_RANDOM_H_
#include <cstdint>

class Random
{
public:
 //generuje liczbe losowa <0:1>
 //o rozkladzie rownomiernym
 //argument numerem generatora
 static double Normal(int i);

 //generuje liczbe losowa <min:max>
 //o rozkladzie rownomiernym
 //argument numerem generatora
 static double Normal(int min, int max, int i);

 //generuje liczbe losowa
 //o rozkladzie wykladniczym
 //argument numerem generatora
 static double Wykladn(int i);
```

```
 //tutaj inicjalizuje seed oraz intensywnosc L
 static void Init(int64_t kernel, double L);

 //testowanie generatorów
 static void Test(int64_t kernel);
 static int64_t KernelNext();

private:
 //jadro generatora, seed
 //6 - TPW
 //5 - losowanie z kolejki do procesora
 //4 - TPG
 //3 - wybor kolejki do procesorow
 //2 - TPO
 //1 - numer urzadzenia
 //0 - SJF
 static int64_t kernel_[7];

 static int64_t kernel_next_;
 static int64_t m_;
 static int64_t a_;

 //L_ musi byc tak dobrane by procesy nie czekaly dluzej niz 50ms
 static double L_;
 static bool test_;
};
#endif
```

random.cpp

```
#include "random.h"
#include "math.h"
#include <fstream>
#include "dane.h"

int64_t Random::kernel_[] = { 0, 0, 0, 0, 0, 0, 0 };
int64_t Random::kernel_next_ = 0;
int64_t Random :: m_ = 2147483647;
int64_t Random :: a_ = 16807;
double Random :: L_ = 1;
bool Random::test_ = 1;

double Random :: Normal(int i)
{
 kernel_[i] = (kernel_[i] * a_) % m_;
 return static_cast<double>(kernel_[i])/m_;
}

double Random :: Normal(int min, int max, int i)
{
 return min + (max-min)*Normal(i);
}

double Random :: Wykladn(int i)
{
 return ((-1 / L_) * log(Normal(i)));
}

void Random :: Init(int64_t kernel, double L)
{
 Dane::SetKernel(kernel);
 Dane::SetL(L);
```

```
 if(test_)
  Test(kernel);
 kernel_[0] = kernel;
 for (int i = 1; i < 7; i++)
 {
  kernel_[i] = kernel_[i - 1];
  for (int j = 0; j < 200000; j++)
   kernel_[i] = (kernel_[i] * a_) % m_;
 }
 kernel_next_ = kernel_[6];
 L_ = L;

 for (int i = 0; i < 200000; i++)
  kernel_next_ = (kernel_next_ * a_) % m_;

 std::string tryb;
 if (Dane::GetNumerSymulacji())
  tryb = "a";
 else tryb = "w";
 FILE* stats;
 stats = fopen("Statystyki.txt", tryb.c_str());
 Dane::SetStats(stats);
 fprintf(stats, "\nKernel: %d   L: %f \n Sredni czas oczekiwania na procesor w kolejnych
iteracjach: ", static_cast<int>(kernel), L);
}

void Random::Test(int64_t kernel)
{
 kernel_[0] = 1117;
 for (int i = 1; i < 7; i++)
 {
  kernel_[i] = kernel_[i - 1];
  for (int j = 0; j < 200000; j++)
   kernel_[i] = (kernel_[i] * a_) % m_;
 }
 L_ = 0.05;
 FILE* normal = fopen("Normal.txt", "w");
 for (int j = 0; j < 7; j++)
 {
  for (int i = 0; i < 10000; i++)
  {
   fprintf(normal, "%f ", Normal(j));
  }
  fprintf(normal, "\n");
 }
 fclose(normal);

 kernel_[0] = 1271;
 L_ = 0.05;
 FILE* wyklad = fopen("Wykladn.txt", "w");
 for (int j = 0; j < 7; j++)
 {
  for (int i = 0; i < 10000; i++)
  {
   fprintf(normal, "%f ", Wykladn(j));
  }
  fprintf(normal, "\n");
 }
 fclose(wyklad);
 test_ = 0;
}

int64_t Random::KernelNext()
{
 return kernel_next_;}
```

system_komputerowy.h

```
#ifndef  SYMULACJA_SYSTEM_KOMPUTEROWY_H_
#define SYMULACJA_SYSTEM_KOMPUTEROWY_H_
#include "procesor.h"
#include "io.h"

//gromadzi kolejki systemu
//z ktorych wybierany jest proces do obslugi przez procesor
//Przyklad uzycia:
//SystemKomputerowy sys();
//sys.DodajProces(p);
class SystemKomputerowy
{
public:
 SystemKomputerowy();
 ~SystemKomputerowy();

 //wprowadza nowy proces do systemu
 //zaplanuj nowy proces TPG (1:30) <-tu nie wiedzialem jaki przedzial,
 //ale przy doborze jak wyzej mozna zaobserwowac zapelnianie kolejek - po dluzszym czasie
 void DodajProces(Proces* proces);

 //zwraca tablice w ktorej umieszamy nasze kolejki k1 i k2
 SJF** KolejkaK();

private:
 SJF** kolejki_k_;
};
#endif
```

system_komputerowy.cpp

```
#include <cstdio>
#include "system_komputerowy.h"

SystemKomputerowy::SystemKomputerowy()
{
 kolejki_k_ = new SJF*[2];
 kolejki_k_[0] = new Kolejka();
 kolejki_k_[1] = new SJF();
}

SystemKomputerowy::~SystemKomputerowy()
{
 delete kolejki_k_[0];
 delete kolejki_k_[1];
 delete[] kolejki_k_;
}

void SystemKomputerowy::DodajProces(Proces* proces)
{
 kolejki_k_[0]->DodajProces(proces);
}

SJF** SystemKomputerowy::KolejkaK()
{
 return kolejki_k_;
}
```

main.cpp

```cpp
#include "model.h"
#include "random.h"
#include "dane.h"
#include <iostream>
#include <string>
#include <fstream>

int main()
{
 bool restart = true;
 while (restart) {
  FILE* do_pliku;
  do_pliku = fopen(Dane::ZmienNazwe().c_str(), "w");
  Dane::SetDoPliku(do_pliku);
  restart = false;
  Model* model_symulacji = new Model();
  model_symulacji->Menu();
  model_symulacji->Wykonaj();
  restart = model_symulacji->Powtorzyc();
  Dane::Reset();
  delete model_symulacji;
  fclose(do_pliku);
  fclose(Dane::GetStats());
 }
 std::system("Pause");
}
```