

# Sprawozdanie 1 – Biblioteka SimEvents i typy kolejek

## 1. Wstęp

Celem tego ćwiczenia było zapoznanie się z biblioteką SimEvents oraz dla zadanego problemu stworzyć prostą aplikację obliczającą czas oczekiwania wiadomości w kolejce, czas opuszczenia serwera przez wiadomość, itp. dla różnych typów kolejek (FIFO, LIFO, SIRO, SJF).

## 2. Wykonanie ćwiczenia

### 2.1. Opisanie stanów systemu

Rozważany system może znajdować się w podanych poniżej stanach (których nazwy świadczą o tym, co się w nich dzieje).

- 1) Czekanie na wiadomość
- 2) Wstawienie nadchodzącej wiadomości do kolejki
- 3) Zdjęcie wiadomości z kolejki
- 4) Przetwarzanie wiadomości
- 5) Opuszczenie serwera przez wiadomość

### 2.2. Pseudokod do wyliczania czasu oczekiwania w kolejce $i$ -tej wiadomości $d_i$ oraz czas opuszczenia serwera przez $i$ -tą wiadomość $c_i$ .

$n$  – liczba wiadomości,

$a_i = [a_1, a_2, \dots, a_n]$  – czasy przybycia każdej wiadomości,

$s_i = [s_1, s_2, \dots, s_n]$  – czasy przetwarzania każdej wiadomości,

$d_i = [d_1, d_2, \dots, d_n]$  – czasy oczekiwania w kolejce  $i$ -tej wiadomości,

$c_i = [c_1, c_2, \dots, c_n]$  – czasy opuszczenia serwera  $i$ -tej wiadomości

$d_1 = 0$

$aa = a$                       % kopia  $a_i$  na którą są nanoszone poprawki związane z czekaniem wiad.

**dla każdej wiadomości  $i$ :**

**jeśli  $a_i + s_i > aa_{i+1}$ , to:**

$aa_{i+1} = aa_i + s_i$

$d_{i+1} = aa_{i+1} - a_{i+1}$

**w przeciwnym razie:**

$aa_{i+1}$  pozostaje bez zmian

$d_{i+1} = 0$

**koniec warunku**

$c_i = aa_i + s_i$

**koniec pętli**

### 2.3. Obliczenie $d_i$ oraz $c_i$ oraz średniego czasu nadchodzenia wiadomości dla podanych danych.

Po wykonaniu wcześniejszego kroku od razu przystąpiłem do implementacji kodu

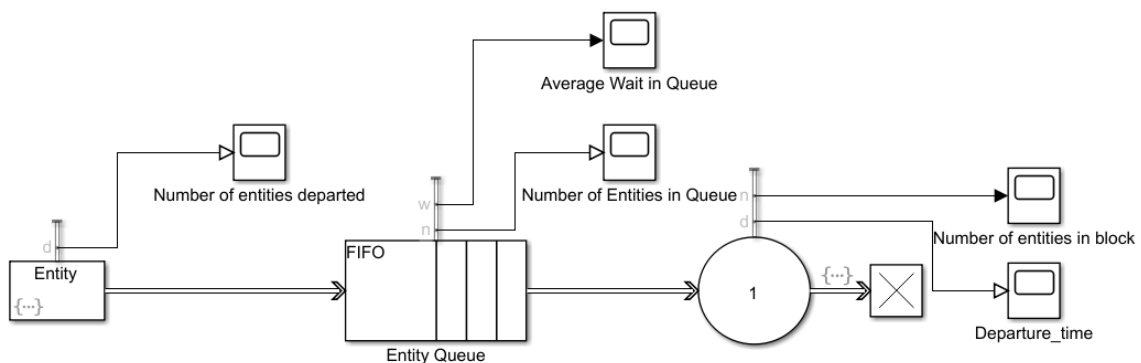
w Matlabie, dzięki czemu łatwo policzyłem wartości dla wszystkich wiadomości.

i	1	2	3	4	5	6	7	8	9	10
$a_i$	4	42	70	100	123	145	190	226	310	322
$s_i$	43	36	34	30	30	40	31	29	36	30
$d_i$	0	5	13	17	24	32	27	22	0	24
$c_i$	47	83	117	147	177	217	248	277	346	376
$\bar{s}_r$	4,0	23	24,7	26	25,4	24,8	27,7	28,8	34,9	32,6

Po przeanalizowaniu kilku początkowych wiadomości można stwierdzić, że są one poprawne

#### 2.4. Schemat omawianego systemu w SimEvents.

Adaptując przykładowy model dostępny we wcześniej wymienionej bibliotece na potrzeby rozważanego problemu utworzyłem widoczny poniżej model symulacyjny, którego zasadniczym zadaniem jest zbieranie danych statystycznych z przeprowadzonej symulacji kolejkowania serwera.



#### 2.5. Utworzenie aplikacji obliczającej czas oczekiwania wiadomości w kolejce, czas opuszczenia serwera przez wiadomość, średni czas nadchodzenia wiadomości, średni czas pozostawiania wiadomości na serwerze oraz średnią długość kolejki.

Stworzona przeze mnie aplikacja opiera się na simulinkowym modelu oraz obliczeniach bazujących na wynikach jego symulacji znajdujących się w m-pliku. Aby symulacja działała poprawnie dla różnych typów kolejek, konieczne było wygenerowanie w bloku *entity* wiadomości z odpowiednim czasem pojawienia się oraz czasu obsługi. Ponadto każda wiadomość, na potrzeby symulacji kolejki SJF, wiadomość otrzymywała priorytet zależny od czasu przetwarzania. Oprócz obsługi kolejki, w modelu znajduje się szereg bloków „scope”, dzięki którym możliwa jest wizualizacja symulacji oraz eksport danych do m-pliku, gdzie z łatwością można te dane przetwarzać. Poniżej zamieszczony jest kod służący do obliczeń.

```
sim('server');
a = [4 42 70 100 123 145 190 226 310 322];
%s = [43 36 34 30 30 40 31 29 36 30];
s = [53 46 44 40 40 50 41 39 46 40];
r = [4 38 28 30 3 22 45 36 84 12];
d = [0 0 0 0 0 0 0 0 0 0];
```

```

c = [0 0 0 0 0 0 0 0 0 0];
w = [0 0 0 0 0 0 0 0 0 0];
for i=1:1:10
    c(order(i)) = dep_time(i,1);
    d(order(i)) = dep_time(i,1)-s(order(i))-a(order(i));
    w(order(i)) = dep_time(i,1) -a(order(i));
end
average_w = mean(w);
average_r = mean(r);
aver_queue_size = aver_queue_size(ceil(c(order(10))/10) + 1,2);

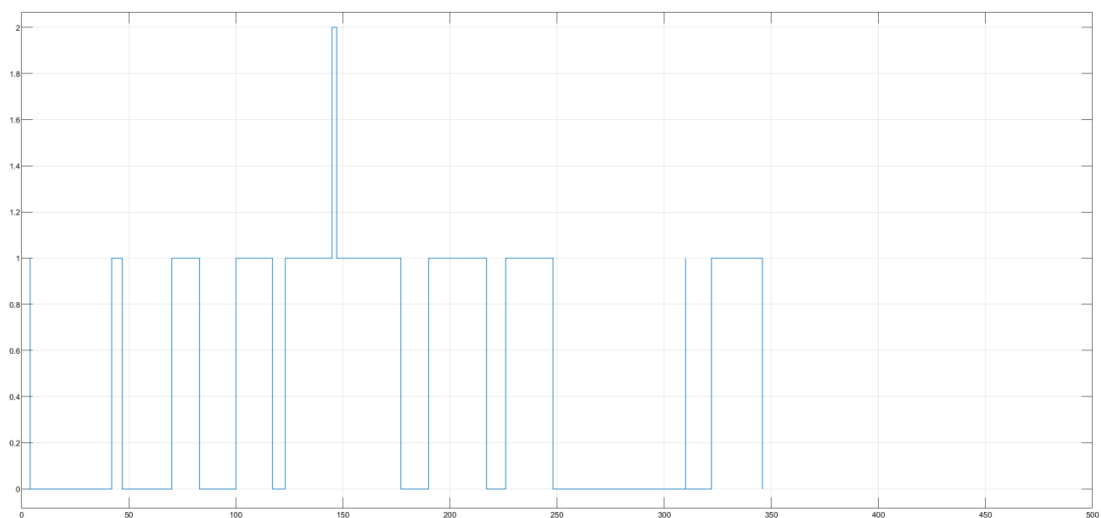
```

### 3. Porównanie wyników

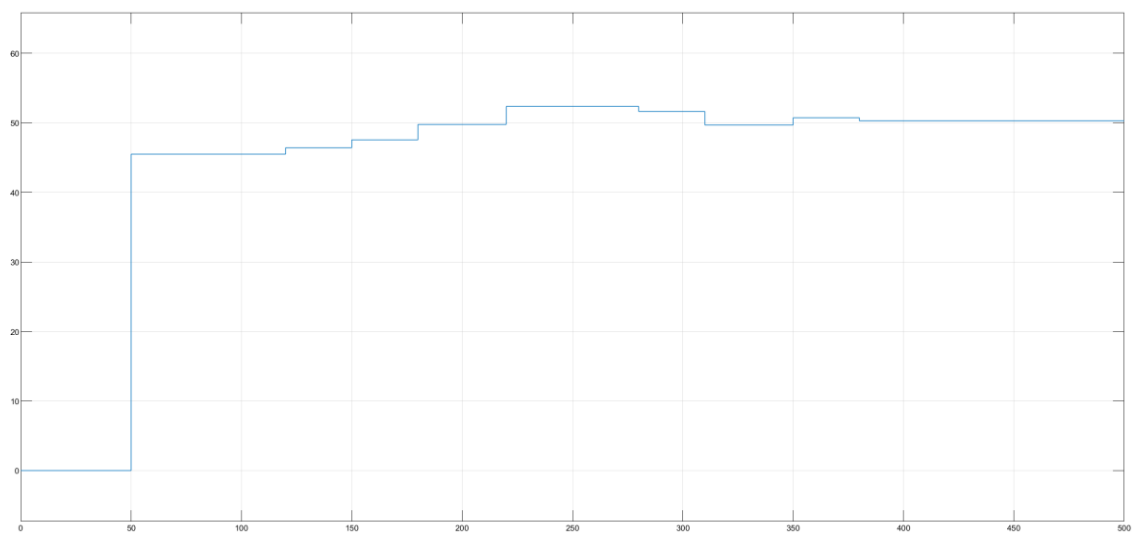
#### 3.1. Kolejka FIFO

Numer wiadomości	1	2	3	4	5	6	7	8	9	10
$a_i$ (czas otrzymania wiadomości)	4	42	70	100	123	145	190	226	310	322
$s_i$ (czas obsługi wiadomości)	43	36	34	30	30	40	31	29	36	30
$c_i$ (czas opuszczenia serwera przez wiadomość)	47	83	117	147	177	217	248	277	346	376
$d_i$ (czas oczekiwania wiadomości w kolejce)	0	5	13	17	24	32	27	22	0	24
$w_i$ (całkowity czas pozostawania na serwerze)	43	41	47	47	54	72	58	51	36	54
Średni czas nadchodzenia wiadomości:	30.2									
Średni czas pozostawania wiadomości na serwerze	50.3									
Średnia długość kolejki	0.4103									

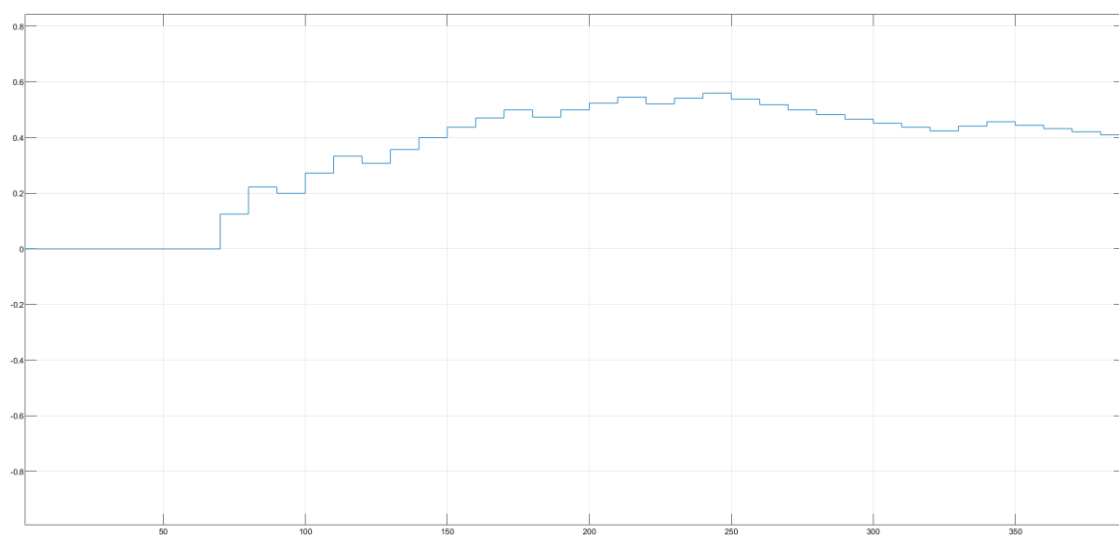
Na poniższym wykresie widać **długość kolejki** w czasie symulacji, na jej podstawie została obliczona **średnia** długość (z użyciem bloczka *mean*).



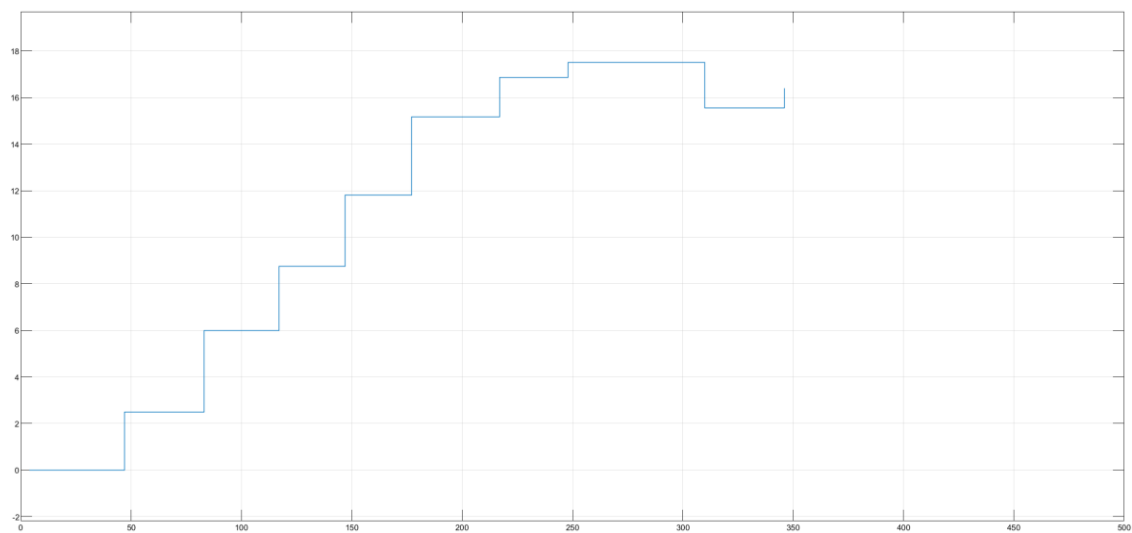
Wykres 1. Kolejka FIFO – ilość wiadomości w kolejce.



**Wykres 2. FIFO -średni czas pozostawiania wiadomości na serwerze (czekanie + przetwarzanie).**



**Wykres 3. FIFO - średnia długość kolejki. Końcowa wartość (380 sekunda) została umieszczona w tabeli.**

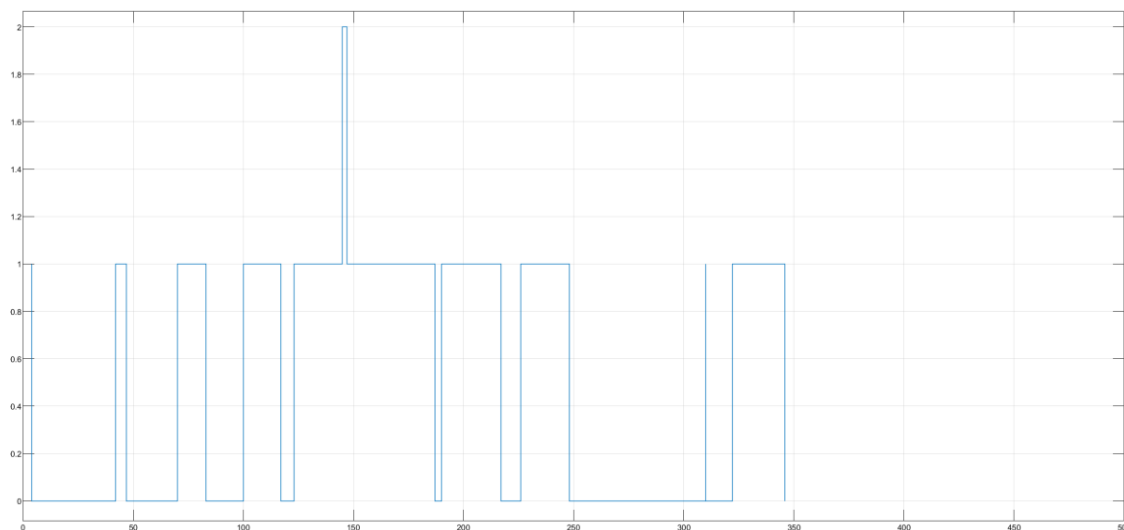


**Wykres 4. FIFO - średni czas oczekiwania wiadomości w kolejce.**

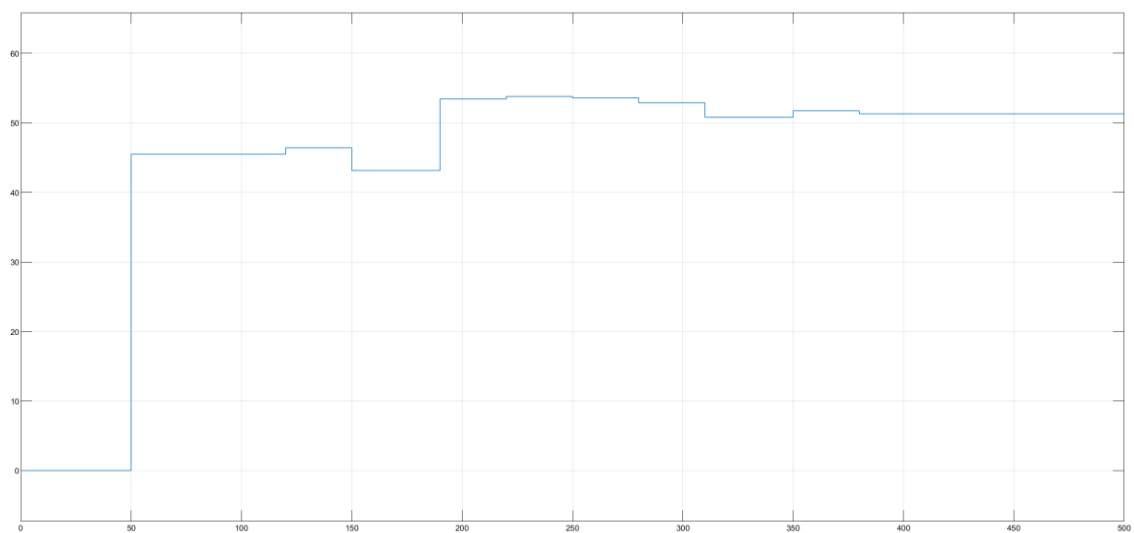
### 3.2. Kolejka LIFO

Numer wiadomości	1	2	3	4	5	6	7	8	9	10
$a_i$ (czas otrzymania wiadomości)	4	42	70	100	123	145	190	226	310	322
$s_i$ (czas obsługi wiadomości)	43	36	34	30	30	40	31	29	36	30
$c_i$ (czas opuszczenia serwera przez wiadomość)	47	83	117	147	217	187	248	277	346	376
$d_i$ (czas oczekiwania wiadomości w kolejce)	0	5	13	17	64	2	27	22	0	24
$w_i$ (całkowity czas pozostawania na serwerze)	43	41	47	47	94	42	58	51	36	54
Średni czas nadchodzenia wiadomości:	30.2									
Średni czas pozostawania wiadomości na serwerze	51.3									
Średnia długość kolejki	0.4359									

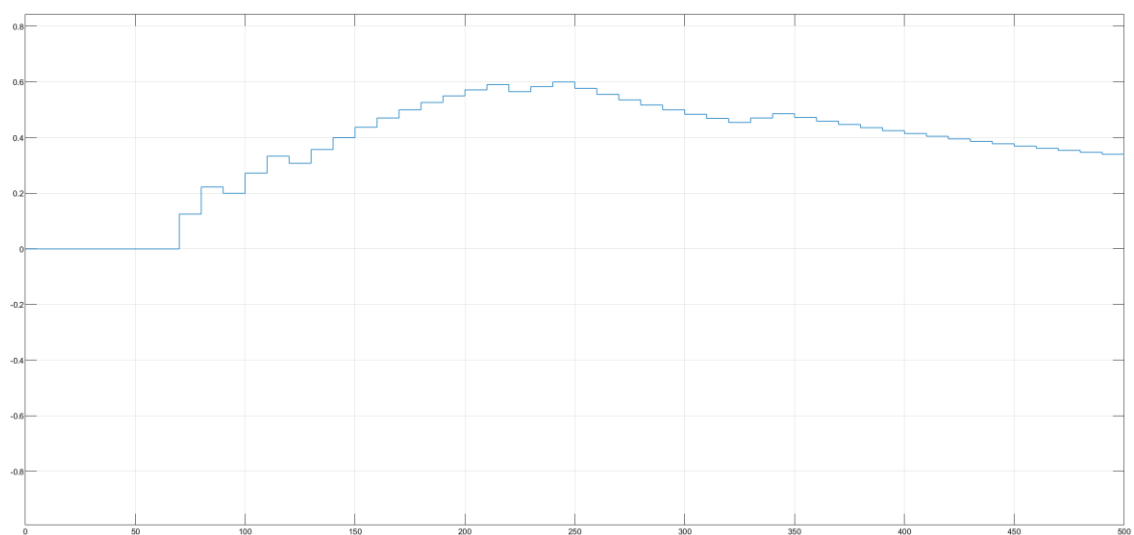
W powyższej tabeli zostały zaznaczone wiadomości, których kolejności przetworzenia zostały zamienione ze względu na zastosowanie kolejki LIFO. Sytuacja taka (zamiana miejsc względem FIFO) mogła wystąpić tylko w sytuacji, kiedy w kolejce są co najmniej 2 elementy do obsłużenia. Na czerwono zostały zaznaczone większe w porównaniu do kolejki FIFO średni czas pozostawania wiadomości na serwerze oraz średnia długość kolejki.



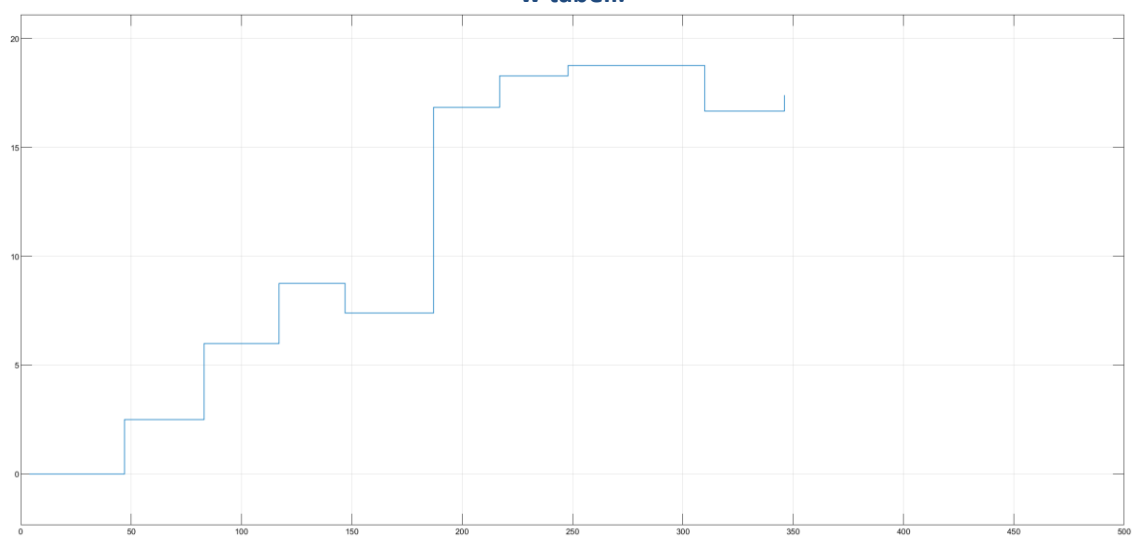
Wykres 5. Kolejka LIFO – ilość wiadomości w kolejce.



**Wykres 6. LIFO - średni czas pozostawania wiadomości na serwerze (czekanie + przetwarzanie).**



**Wykres 7. LIFO - średnia długość kolejki. Końcowa wartość (380 sekunda) została umieszczona w tabeli.**



**Wykres 8. LIFO - średni czas oczekiwania wiadomości w kolejce.**

### 3.3. Kolejka SJF

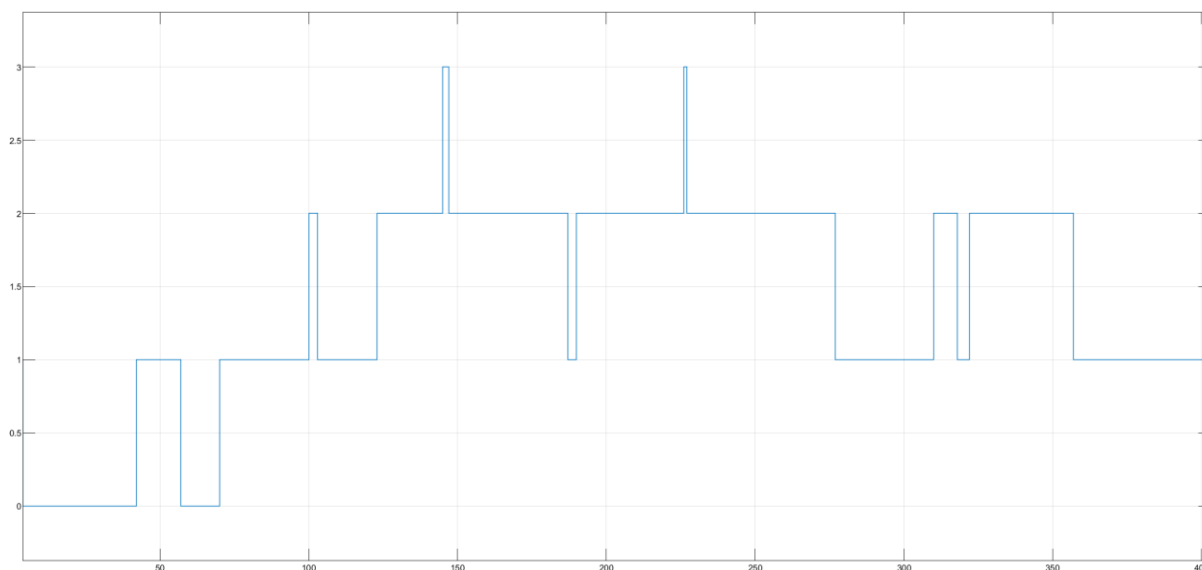
W kolejce tej każda wiadomość ma predefiniowany priorytet, który może być przeskalowany liniowo od najdłuższego czasu wykonania do najkrótszego. Inną opcją jest przypisanie wartości całkowitej priorytetu dla każdej wiadomości. Ja w swojej symulacji zdecydowałem się na tą drugą opcję ze względu na to, że wiadomości jest niewiele i z łatwością można je uszeregować według rosnącego czasu przetwarzania. Aby ustawić wiadomościom priorytety, skorzystałem z atrybutu `PriorityA` w bloku *entity generator*.

```
PRIO = [4 6 7 9 9 5 8 9 6 10];  
(...)  
entity.PriorityA = PRIO(idx);
```

Numer wiadomości	1	2	3	4	5	6	7	8	9	10
$a_i$ (czas otrzymania wiadomości)	4	42	70	100	123	145	190	226	310	322
$s_i$ (czas obsługi wiadomości)	43	36	34	30	30	40	31	29	36	30
$c_i$ (czas opuszczenia serwera przez wiadomość)	47	83	117	147	177	217	248	277	346	376
$d_i$ (czas oczekiwania wiadomości w kolejce)	0	5	13	17	24	32	27	22	0	24
$w_i$ (całkowity czas pozostawania na serwerze)	43	41	47	47	54	72	58	51	36	54
Średni czas nadchodzenia wiadomości:	30.2									
Średni czas pozostawania wiadomości na serwerze	50.3									
Średnia długość kolejki	0.4103									

Łatwo można zauważyć, że wartości w tabeli niczym się nie różnią od kolejki **FIFO**. Dzieje się tak, ponieważ tylko raz w kolejce są dwie wiadomości, między którymi należy rozstrzygnąć, która będzie przetworzona wcześniej. Jako że wyższy priorytet akurat w tym zestawie danych ma w tamtej sytuacji wiadomość o niższym indeksie, to kolejka jest identyczna do **FIFO**. Gdyby dłuższy czas obsługi oznaczał większy priorytet, to kolejka działałaby dla tego zestawu danych identycznie do **LIFO**.

Zamieszczanie wykresów z tej symulacji uznałem za zbędne, ponieważ są identyczne wyżej. Zamiast tego zmodyfikowałem nieco dane wejściowe tak, aby konieczność rozstrzygania pierwszeństwa wiadomości była częściej. W tym celu do czasu przetwarzania każdej wiadomości dodałem 10 i przeprowadziłem symulację.



Wykres 9. Kolejka SJF na zmienionych danych – liczba wiadomości w kolejce.

Widać na powyższym wykresie, że zamierzony cel został osiągnięty. Potrzeba rozstrzygnięcia pierwszeństwa wystąpiła 7 razy. Sprawdźmy zatem poprawność i co ważniejsze – efektywność implementacji tej kolejki.

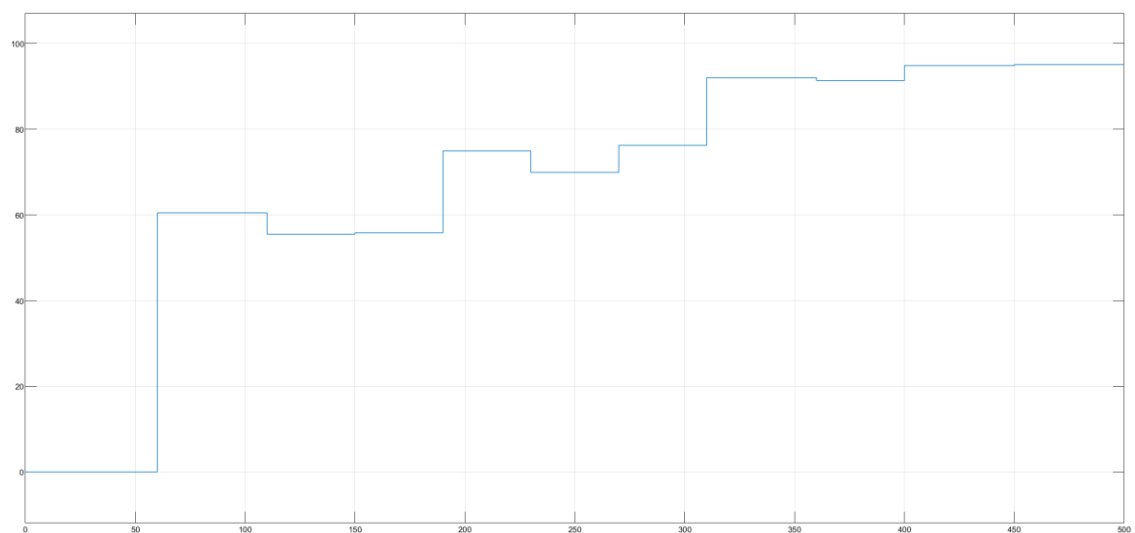
Numer wiadomości	1	2	3	4	5	6	7	8	9	10
$a_i$ (czas otrzymania wiadomości)	4	42	70	100	123	145	190	226	310	322
$s_i$ (czas obsługi wiadomości)	53	46	44	40	40	50	41	39	46	40
priorytet wiadomości	4	6	7	9	9	5	8	10	6	9
$c_i$ (czas opuszczenia serwera przez wiadomość)	57	103	227	143	183	357	307	266	443	397
$d_i$ (czas oczekiwania wiadomości w kolejce)	0	15	113	3	20	162	76	1	87	35
$w_i$ (całkowity czas pozostawania na serwerze)	53	61	157	43	60	212	117	40	133	75
Średni czas nadchodzenia wiadomości:	30.2									
Średni czas pozostawania wiadomości na serwerze	95.1									
Średnia długość kolejki	1.1304									

Widać, że wiadomości zostały obsłużone w następującej kolejności:

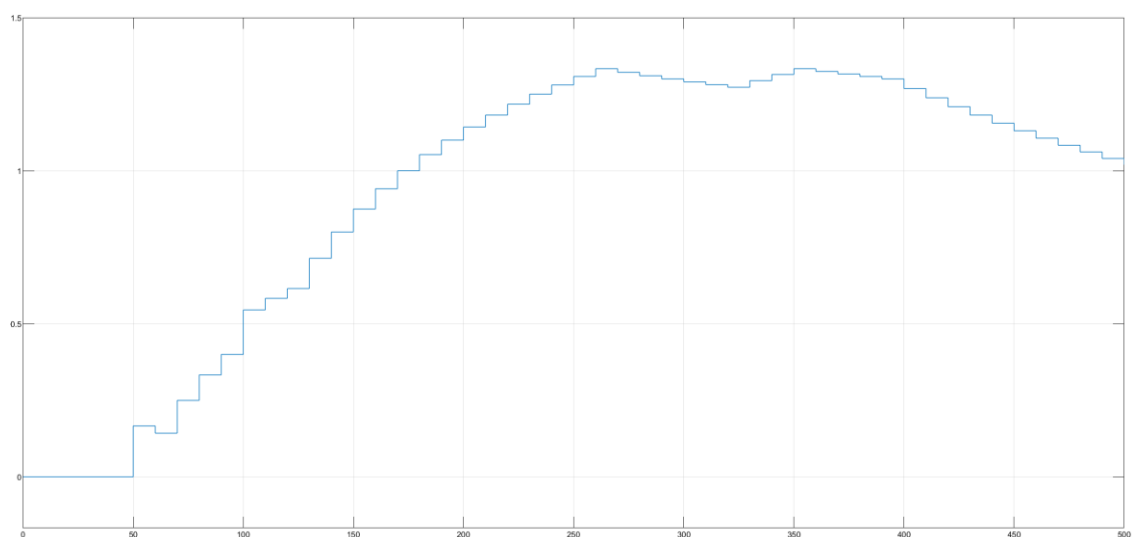
1 -> 2 -> 4 -> 5 -> 3 -> 8 -> 7 -> 6 -> 10 -> 9

Poświęcając chwilę na analizę tej kolejności można stwierdzić, że kolejka prawidłowo przetwarza najpierw najkrótsze zadania. Efektywność takiego zabiegu sprawdzę później.

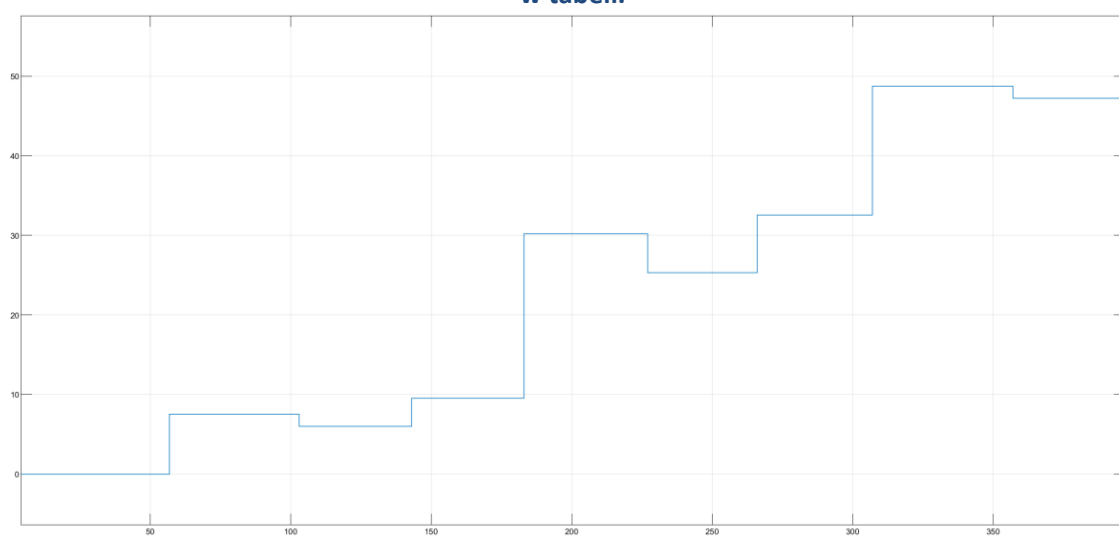




**Wykres 10. SJF - średni czas pozostawiania wiadomości na serwerze (czekanie + przetwarzanie).**



**Wykres 11. SJF - średnia długość kolejki. Końcowa wartość (450 sekunda) została umieszczona w tabeli.**



**Wykres 12. SJF - średni czas oczekiwania wiadomości w kolejce.**

Na poniższych wykresach widać, że średni czas pozostawania wiadomości na serwerze oraz średni czas oczekiwania wiadomości w kolejce mają tendencje wzrostowe. Jest to spowodowane dłuższymi czasami przetwarzania każdej wiadomości niż interwały pomiędzy ich przychodzeniem. Jest to charakterystyczne dla wprowadzonego zestawu danych i dla innych może być zupełnie inaczej.

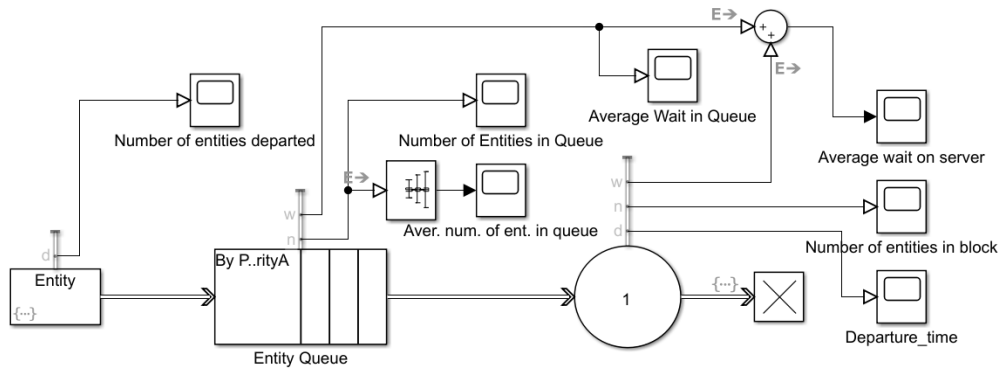
Dla takich danych uruchomiłem ponownie symulację z kolejkami **FIFO** i **LIFO** (dodałem też kolejkę **LJF** – Longest Job First) celem próby porównania kolejek względem przyjętych wskaźników, jakimi są średnia długość kolejki i średni czas oczekiwania wiadomości w kolejce (korelujący z czasem pozostawania na serwerze).

Typ kolejki	LIFO	FIFO	SJF	LJF
Średnia długość kolejki	1.1957	1.1957	1.1304	1.2609
Średni czas oczekiwania wiadomości w kolejce	53.4	54.8	51.2	57.7

## 4. Wnioski

- Na podstawie danych wejściowych do symulacji ( $a_i$ ,  $s_i$ ) trudno jednoznacznie stwierdzić, która kolejka sprawdza się najlepiej, gdyż różnica między **FIFO** i **LIFO** jest w jednym elemencie, natomiast **SJF** w ogóle nie wprowadza nic nowego.
- Wymieniona wcześniej różnica sprawiła, że kolejka **FIFO** okazała się minimalnie lepsza (pod kątem przyjętych wskaźników optymalności). Jednak jest to różnica niewielka i raczej o niczym nie świadczy.
- Po przeprowadzeniu symulacji na nieco bardziej obciążających kolejkę danych można zauważyć, że najlepiej sprawdziła się kolejka **SJF**, a najgorzej **LJF**.
- Trudność jednoznacznej oceny efektywności różnych typów kolejek nie czyni modelu bezużytecznym. W przypadku palącej potrzeby rozstrzygnięcia, którego typu kolejki należy użyć do rzeczywistego modelu, można użyć wartości charakterystycznych dla danego obiektu i wtedy liczyć na satysfakcjonujące wyniki.
- Biblioteka SimEvents pozwala na łatwe tworzenie modeli zdarzeń dyskretnych i symulację tychże zdarzeń. Wykonany model w połączeniu z matlabowskim skryptem pozwala na „wyciągnięcie” bardzo wielu danych statystycznych i innych, pozwalających na ocenę modelu samego w sobie, zastosowanej kolejki i cech symulowanego zdarzenia.

## 5. Pełny model, kod Matlab'a i ważniejsze ustawienia bloków



Block Parameters: Entity Generator

Entity Generator

Generate entities using intergeneration times from dialog or upon arrival of events. Optionally, specify entity types as anonymous, structured, or bus.

Entity generation Entity type Event actions Statistics

Generation method: Time-based

Time source: MATLAB action

Intergeneration time action:

```

1 % Pattern: Repeating Sequence
2 persistent A;
3 persistent idx;
4 if isempty(A)
5     A = [4 38 28 30 23 22 45 36 84 12];
6     idx = 1;
7 end
8 if idx > numel(A)
9     dt = inf;
10 else
11     dt = A(idx);
12 end
13 idx = idx + 1;

```

☐ Generate entity at simulation start

Insert pattern ...

OK Cancel Help Apply

Block Parameters: Entity Generator

Entity Generator

Generate entities using intergeneration times from dialog or upon arrival of events. Optionally, specify entity types as anonymous, structured, or bus.

Entity generation Entity type Event actions Statistics

Event actions

Generate action:

Called after entity is generated.  
To access attribute use: entity.Number

```

1 % Pattern: Repeating Sequence
2 persistent A;
3 persistent S;
4 persistent PRIO;
5 persistent idx;
6 if isempty(A)
7     A = [1 2 3 4 5 6 7 8 9 10];
8     S = [53 46 44 40 40 50 41 39 46 40];
9     PRIO = [4 6 7 9 9 5 8 9 6 10];
10    idx = 1;
11 end
12 if idx > numel(A)
13     entity.Number = inf;
14     entity.PriorityA = inf;
15     entity.Durations = inf;
16 else
17     entity.Number = A(idx);
18     entity.PriorityA = PRIO(idx);
19     entity.Durations = S(idx);
20 end
21 coder.extrinsic('assignin');
22 idx = idx + 1;

```

Entity structure

- entity
  - Number
  - PriorityA
  - Durations
- entitySys
  - id
  - priority

Insert pattern ...

OK Cancel Help Apply

Block Parameters: Entity Queue

Queue

Store messages or entities in a queue. The block can queue items based on arrival order or priority. The item at the head of the queue departs when the downstream block is ready to accept it. You can specify the queue capacity.

Main Event actions Statistics

☐ Overwrite the oldest element if queue is full

Capacity: inf

Queue type: Priority

Priority source: PriorityA

Sorting direction: Descending

Entity arrival source: Input port

OK Cancel Help Apply

Block Parameters: Entity Server

Entity Server

Serve multiple entities independently for a period of time and then attempt to output each entity through the output port. If the output port is blocked, the pending entity stays in this block until the port becomes unblocked. You can specify the service time, which is the duration of service, via a parameter, attribute, or signal.

When the block permits preemption, an entity in the server can depart early through a second port.

Main Event actions Preemption Statistics

Capacity: 1

Service time source: Attribute

Service time attribute name: Durations

OK Cancel Help Apply

Block Parameters: Entity Terminator

Accept and destroy entities.

Event actions Statistics

Event actions

Entry action:

Called after entity has entered this block.  
To access attribute use: entity.Number

```

1 persistent idx;
2 persistent A;
3 coder.extrinsic('assignin');
4 if isempty(A)
5     A = [0 0 0 0 0 0 0 0 0 0];
6     idx = 1;
7 end
8 A(idx)=entity.Number;
9 if idx == 10
10    assignin('base', 'order', A);
11 end
12 idx = idx+1;

```

Entity structure

- entity
  - Number
  - PriorityA
  - Durations
- entitySys
  - id
  - priority

Insert pattern ...

OK Cancel Help Apply

```

%% zad2 obliczanie czasow (tylko dla FIFO)
clear all; close all;
n = 10;
a = [4 42 70 100 123 145 190 226 310 322];
s = [43 36 34 30 30 40 31 29 36 30];
a_copy = a;

d = (0);
c = (0);
for i = 1:n-1
    if (a_copy(i)+s(i) > a_copy(i+1))
        a_copy(i+1) = a_copy(i)+s(i);
        d(i+1) = a_copy(i+1)-a(i+1);
    else
        a_copy(i+1) = a_copy(i+1);
        d(i+1) = 0;
    end
    c(i) = a_copy(i) + s(i);
end
c(10) = a_copy(10) + s(10);

%% zad5 wyliczanie danych dla roznych rodzajow kolejek
clear all; close all;
sim('server');
a = [4 42 70 100 123 145 190 226 310 322];
% s = [43 36 34 30 30 40 31 29 36 30];
s = [53 46 44 40 40 50 41 39 46 40];
r = [4 38 28 30 3 22 45 36 84 12];
d = [0 0 0 0 0 0 0 0 0 0];
c = [0 0 0 0 0 0 0 0 0 0];
w = [0 0 0 0 0 0 0 0 0 0];
for i=1:1:10
    c(order(i)) = dep_time(i,1);
    d(order(i)) = dep_time(i,1)-s(order(i))-a(order(i));
    w(order(i)) = dep_time(i,1) -a(order(i));
end
average_w = mean(w);
average_r = mean(r);
aver_queue_size = aver_queue_size(ceil(c(order(10))/10) + 1,2);

```