



УНИВЕРЗИТЕТ У НОВОМ САДУ  
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У  
НОВОМ САДУ



### ***Pub/Sub сервис***

Индустријски и комуникациони протоколи у  
електроенергетским системима  
- Примењено софтверско инжењерство (ОАС) -

Нови Сад, 29.01.2021.

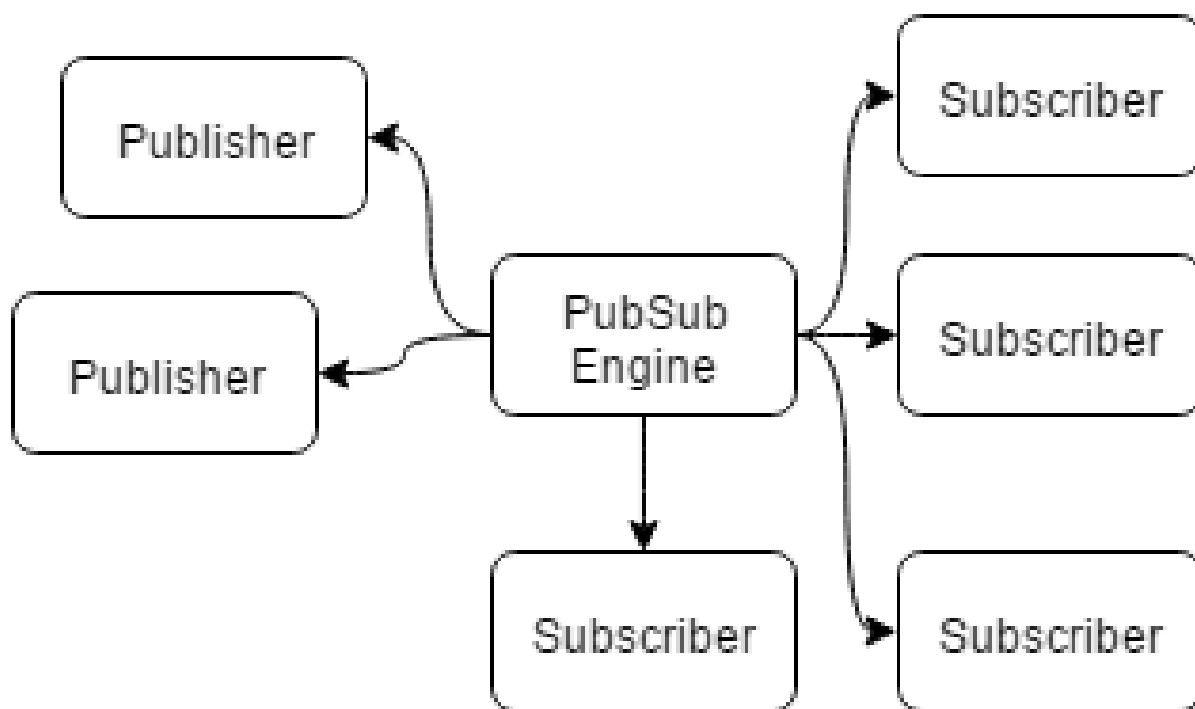
## САДРЖАЈ

1. УВОД
2. ДИЗАЈН
3. СТРУКТУРЕ ПОДАКА
4. РЕЗУЛТАТИ ТЕСТИРАЊА
5. ЗАКЉУЧАК
6. ПОТЕНЦИЈАЛНА УНАПРЕЂЕЊА

## 1. Увод

У софтверској архитектури, *publish-subscribe* је образац за размену порука код којег пошиљаоци порука, названи *publisher*-и, не програмирају поруке које ће се директно слати одређеним примаоцима, названим *subscriber*-и, већ објављене поруке категоризују у класе без знања који претплатници, ако их има, могу постојати. Слично томе, претплатници изражавају интересовање за једну или више тема (*topic*) и примају само поруке које их занимају, без знања који издавачи, ако их има, могу постојати.

Већина система за размену порука подржавају *Pub/Sub* образац у свом *API*-ју (нпр. *Java Message Service*). Овај образац пружа већу скалабилност мреже и динамичнију топологију мреже, што резултира смањеном флексибилношћу за модификовање издавача и структуре објављених података. *Publish/Subscribe (Pub/Sub)* порука пружа тренутна обавештења о догађајима за ове дистрибуиране апликације. *Publish Subscribe* модел омогућава асинхронно емитовање порука у различите делове система. *Topic* поруке пружа лагани механизам за емитовање асинхронних обавештења о догађајима, као и крајње тачке (*endpoints*) које омогућавају компонентама система (*Publisher*, *Subscriber*) да се повежу са одговарајућим *Topic*-ом како би слале и примале те поруке.



Слика 1. Архитектура *Pub/Sub* сервиса

## 2. Дизајн

У апликацији су дефинисана три кључне компоненте:

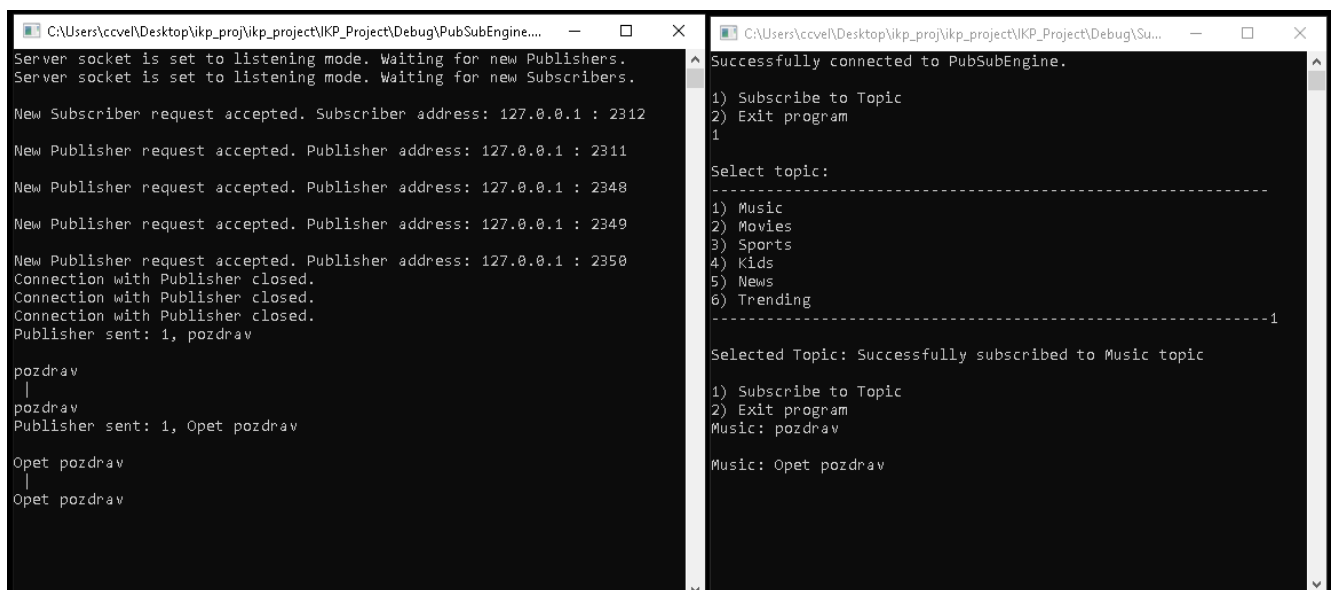
- 1) *Publisher*
- 2) *Subscriber*
- 3) *PubSubEngine*

*Publisher* генерише жељени садржај, повезује се на познату адресу *PubSubEngine*-а помоћу *Socket*-а, па након тога тај садржај смешта у одређену структуру података жељеног *Topic*-а (*Publish to Topic*).

Повезивање на *PubSubEngine* као и смештање порука у структуру одговарајућег *Topic*-а имплементирано је употребом нити (*Thread*) која константно ослушкује нове конекције и потом смешта пристигли садржај у ту структуру.

*PubSubEngine* тај садржај даље прослеђује у нит (*Thread*) који ће за сваког појединачног претплатника, из листе претплаћених, креирани посебну нит (*Thread*) која ће му тај садржај проследити.

Претплатник (*Subscriber*) се претплаћује на *Topic*-е које одабере (*Subscribe to Topic*), и на тај начин даје до знања *PubSubEngine*-у да жели да буде обавештен о сваком новом садржају који претходно стигне на тај *Topic*. Претплатом на одређени *Topic* омогућује да *PubSubEngine* може да га обавести ако *Publisher* изда нови садржај на *Topic* на који је претплаћен. Тиме бива додан у листу свих претплатника на *PubSubEngine*-у, као и у листу претплатника у оквиру тог *Topic*-а.



```
C:\Users\ccvef\Desktop\vikp_proj\vikp_project\IKP_Project\Debug\PubSubEngine....
Server socket is set to listening mode. Waiting for new Publishers.
Server socket is set to listening mode. Waiting for new Subscribers.

New Subscriber request accepted. Subscriber address: 127.0.0.1 : 2312
New Publisher request accepted. Publisher address: 127.0.0.1 : 2311
New Publisher request accepted. Publisher address: 127.0.0.1 : 2348
New Publisher request accepted. Publisher address: 127.0.0.1 : 2349
New Publisher request accepted. Publisher address: 127.0.0.1 : 2350
Connection with Publisher closed.
Connection with Publisher closed.
Connection with Publisher closed.
Publisher sent: 1, pozdrav

pozdrav
|
pozdrav
Publisher sent: 1, Opet pozdrav

Opet pozdrav
|
Opet pozdrav

C:\Users\ccvef\Desktop\vikp_proj\vikp_project\IKP_Project\Debug\Su...
Successfully connected to PubSubEngine.

1) Subscribe to Topic
2) Exit program
1

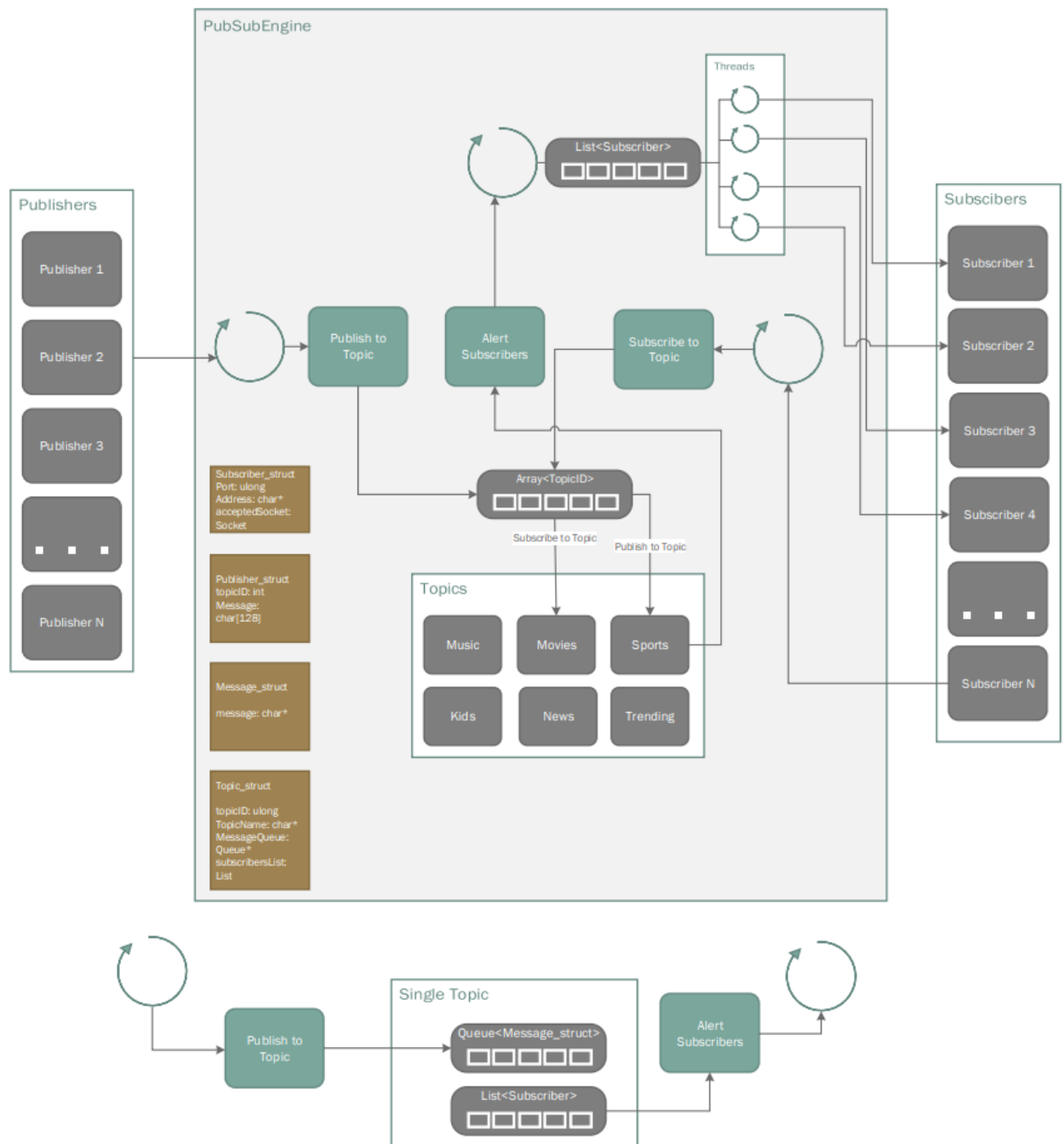
Select topic:
-----
1) Music
2) Movies
3) Sports
4) Kids
5) News
6) Trending
-----1

Selected Topic: Successfully subscribed to Music topic

1) Subscribe to Topic
2) Exit program
Music: pozdrav

Music: Opet pozdrav
```

Слика 2. *PubSubEngine* и *Subscriber* након што *Publisher* објави садржај



Слика 3. Дизајн архитектуре система

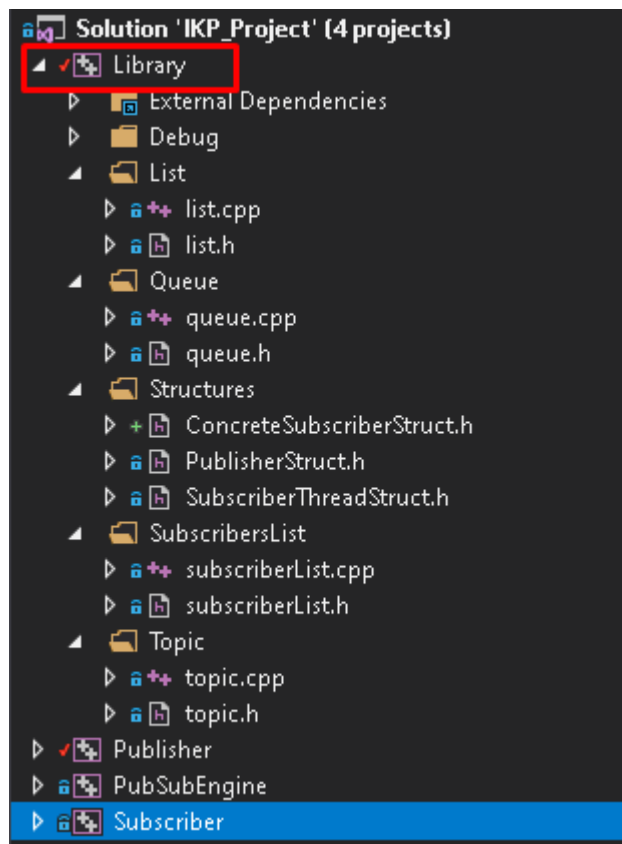
### 3. Структуре података

Новопристигли садржај који одређени *Publisher* издаје на *Topic* се смешта у структуру података тог *Topic*-а – *Queue*.

*Queue* представља *FIFO* (*First-in-first-out*) структуру података. Његовим елементима приступамо истим редоследом којим се они чувају у колекцију. Овакав приступ је погодан код привременог чувања података што, као и сам *FIFO* приступ, у потпуности одговара потребама ове апликације. Поред *Queue* структуре, за чување садржаја *Publisher*-а *Topic* поседује још и листу у којој чува *Subscriber*-е који су на њега претплаћени.

*List* је структура података у којој су елементи повезани помоћу показивача. Чвор представља елемент на повезаној листи који има неке податке и показивач који показује на следећи чвор у листи. Листа је динамичка структура података, тако да може расти и смањивати се током извршавања додељивањем и уклањањем меморије (стога нема потребе за предефинисањем величине листе). Величина листе се може повећавати или смањивати током рада апликације, тако да нема губитка меморије.

У случају низа, неопходно је декларисати његову величину. Ипак, уколико би био дефинисан низ са одређеном величином, и у њега смештено елемената мање од задате величине, долази до непотребног расипања меморије. У овој апликацији, низ је коришћен као структура за смештање *Topic*-а, јер је већ унапред познат њихов број. У овом случају, предност низа у односу на листу је у томе што листа захтева више меморије за чување елемената, јер сваки чвор садржи показивач на следећи, и он захтева додатну меморију за себе. Такође, код листе није могуће насумично приступити чвору на положају  $n$ , већ је неопходно прећи све чворове пре њега (стога је време потребно за приступ чвору велико).



Слика 4. *Library* пројекат са структурама

```

struct listitem {
    int data;
    struct listitem *next;
};

typedef struct listitem Listitem;
struct list {
    Listitem *head;
};

```

Слика 5. *List* структура

```

/* a link in the queue, holds the info and point to the next Node*/
typedef struct {
    char message[256];
} DATA;

typedef struct Node_t {
    DATA data;
    struct Node_t *prev;
} NODE;

/* the HEAD of the Queue, hold the amount of node's that are in the queue*/
typedef struct Queue {
    NODE *head;
    NODE *tail;
    int size;
    int limit;
} Queue;

```

Слика 6. *Queue* структура

```

typedef struct {
    unsigned long topicID;
    const char* TopicName;
    Queue* messageQueue;
    List subscribersList;
} TOPIC;

TOPIC * initTopic();
void storeMessageToQueue(TOPIC* t, char* message, NODE *pN);

```

Слика 7. *Topic* структура

```

typedef struct subscriber_t {
    unsigned short port;
    char* address;
    SOCKET acceptedSocket;
}SUBSCRIBER;

struct listitem {
    SUBSCRIBER subscriber;
    struct listitem *next;
};

typedef struct listitem Listitem;
struct list {
    Listitem *head;
};

```

Слика 8. *Subscriber* структура

```

typedef struct publisher_t {
    int topicID;
    char message[128];
}PUBLISHER;

```

Слика 9. *Publisher* структура

```

typedef struct concreteSubscriberThreadStruct_t {
    char message[BUFFER_SIZE];
    SOCKET acceptedSocket;
}CONCRETE_SUBSCIBER;

```

Слика 10. *Concrete\_Subscriber* структура

```

typedef struct subscriberThreadStruct_t {
    unsigned short port;
    char* address;
    SOCKET acceptedSocket;
}SUBSCRIBER_THREAD;

```

Слика 11. *Subscriber\_thread* структура

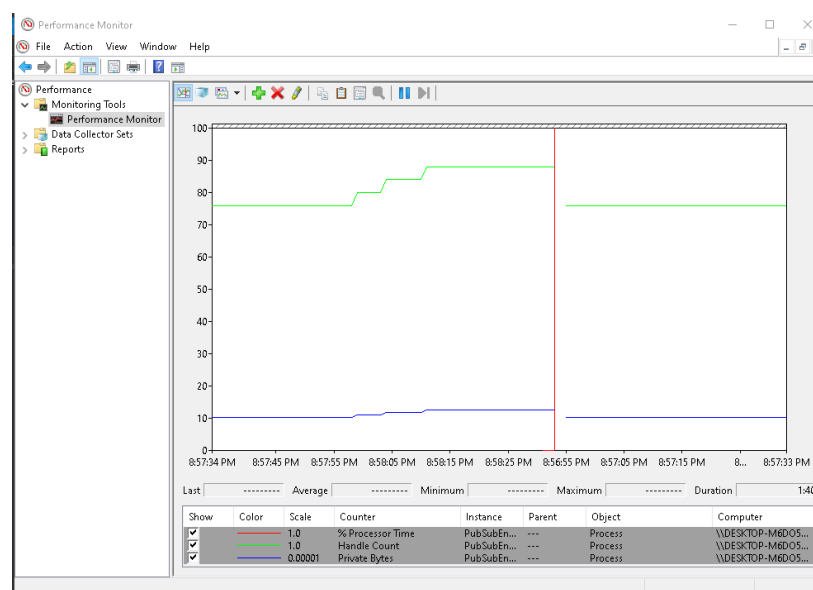


## 4. Резултати тестирања

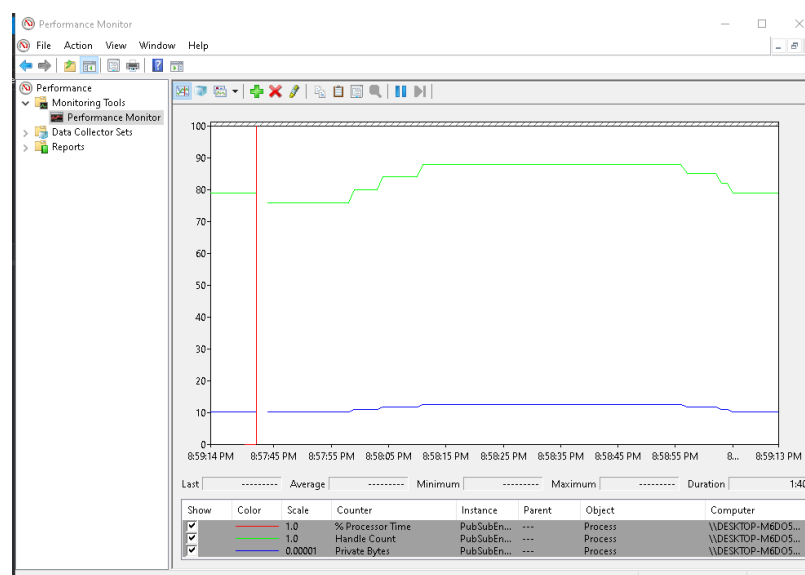
На слици 12 приказана је употреба меморије у тестном сценарију апликације. Приликом конектовања *Publisher*-а на *PubSubEngine*, меморијско заузеће скочи за одређену вредност. Гашењем *Publisher*-а, он је дисконектован са *PubSubEngine*-а, и меморијско заузеће се врати за исту количину која је била заузета приликом конектовања.

ID	Time	Allocations (Diff)	Heap Size (Diff)
1	10.69s	273 (n/a)	88.98 KB (n/a)
2	35.18s	281 (+8 ↑)	91.29 KB (+2.31 KB ↑)
3	48.23s	273 (-8 ↓)	88.98 KB (-2.31 KB ↓)

Слика 12. *Memory Usage* након конектовања/дисконектовања *Publisher*-а



Слика 13. *Performance Monitor* пре гашења *Publisher*-а



Слика 14. *Performance Monitor* након гашења *Publisher*-а

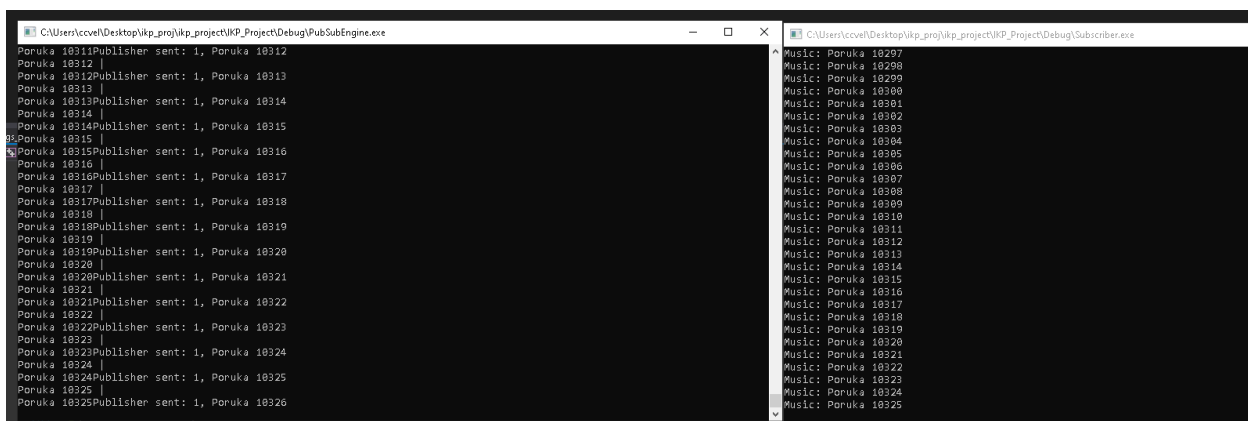
Следећи случај јесте конектовање *Subscriber*-а на *PubSubEngine* и његово дисконектовање са истог. На слици 15 се може уочити да се меморијско заузеће приликом дисконектовања такође ослобађа.

ID	Time	Allocations (Diff)	Heap Size (Diff)
1	10.69s	273 (n/a)	88.98 KB (n/a)
4	97.67s	277 (+4 ↑)	89.76 KB (+0.78 KB ↑)
5	106.63s	269 (-8 ↓)	87.45 KB (-2.31 KB ↓)

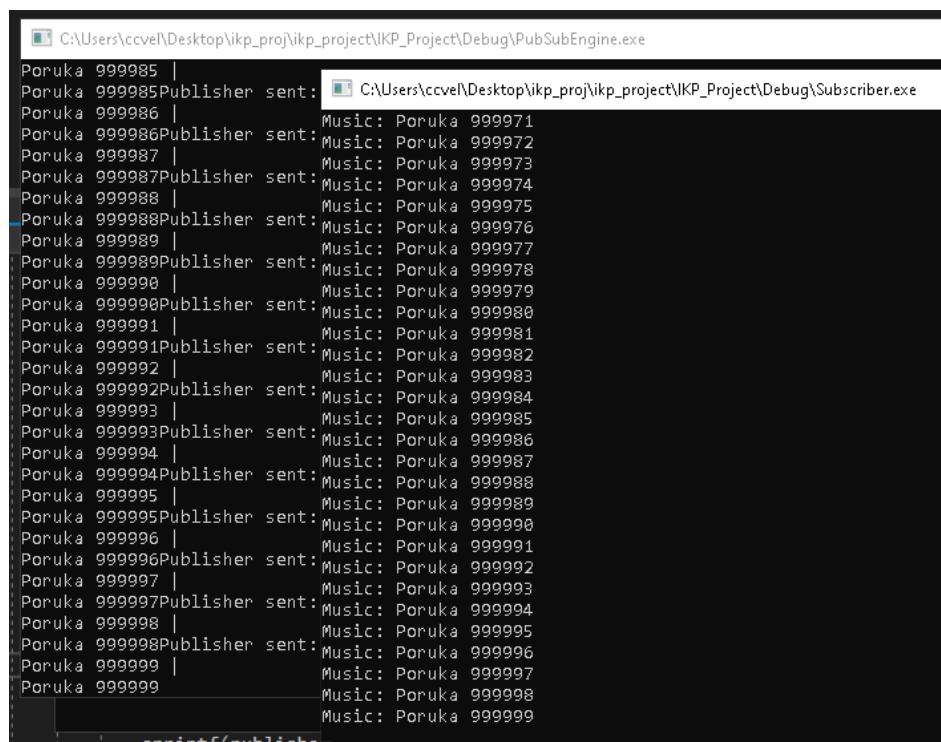
Слика 15. *Memory Usage* након конектовања/дисконектовања *Publisher*-а

За извођење *Stress test*-a, покренут је *Publisher* који поседује и опцију слања одређеног броја порука *PubSubEngine*-у. За конкретан сценарио одабрано је слање десет хиљада порука и милион порука, и смештање истих у *Queue* структуру одређеног *Topic*-a. Приказ слања десет хиљада и милион порука приказан је на сликама 16 и 17.

Резултати теста били би квалитетнији да је постојао сценарио покретања и гашења апликација, али пошто је у овој апликацији предвиђено да *PubSubEngine* сервис константно ради и обрађује захтеве од стране клијената, овакво нешто није могуће приказати употребом датих тест-евиденција.



Слика 16. *Publisher* шаље десет хиљада порука



Слика 17. *Publisher* шаље милион порука

## 5. Закључак

Апликација се у целини понаша спрам дизајнираног сценарија рада. Сва меморијска заузећа успешно се отпуштају, *Thread Handle*-ови се терминирају након што су задовољили услов због ког су креирани. Апликација је способна да опслужује истовремено већи број корисника.

Један од проблема јесте тај што у току активног рада апликације рачунарска меморија може представљати ограничавајући фактор, јер број заузећа одређених структура на *heap*-у зависи управо од меморије која је на располагању кориснику овог система. Пример за то био би *stress test* који је извршен слањем велике количине порука ка *PubSubEngine*-у.

Може се закључити да апликација представља солидну основу за неки скромнији вишекориснички режим рада где су демонстрирани концепти како мрежног програмирања, тако и конкурентности у раду.

## 6. Потенцијална унапређења

У овом поглављу овог истраживања размотрена су потенцијална побољшања која би довела до бољег и лакшег коришћења овог система.

Могуће унапређење рада ове апликације била би употреба оптималнијих структура података за одређене сегменте. У том случају, за смештање *Topic*-а била би коришћена *Hashmap* структура података. Пошто је унапред познат њихов број, могуће их је кроз код филтрирати помоћу добијеног *ID*-а и приступити одређеном члану низа комплексношћу  $O(1)$ . Уколико би број *Topic*-а био варијабилан, тада би подесније било користити речник као структуру јер би у њега био могућ приступ, као и додавање и брисање одређених елемената са  $O(1)$  комплексношћу, али би структура била динамичка, док низ није подложен изменама и ради конкретно за сценарио где је број *Topic*-а предефинисан.