Міністерство освіти і науки України
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
Фізико-технічний інститут

Операційні системи

Лабораторна №8

Виконав:
Студент групи ФБ-82
Козачок Вячеслав
Перевірив:
Кіреєнко О.В.

Київ - 2020

1 Розминка.

Стандартна задача виробник-споживач. C++

1.1 Input data

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Nulla facilisi nullam vehicula ipsum a arcu cursus vitae. Donec et odio pellentesque diam volutpat commodo. In pellentesque massa
placerat duis ultricies lacus sed. Enim nec dui nunc mattis enim. Id consectetur purus ut faucibus pulvinar elementum integer. Pharetra pharetra massa massa ultricies. Pretium quam vulputate dignissim suspendisse in est ante. Sapien nec sagittis aliquam malesuada bibendum. Quis hendrerit dolor magna eget est lorem ipsum. Amet cursus sit amet dictum sit.
Leo integer malesuada nunc vel risus commodo. Arcu non odio euismod lacinia. Orci ac auctor augue mauris. Eu non diam phasellus vestibulum lorem. Nibh tortor id aliquet lectus proin nibh.
Proin fermentum leo vel orci porta non pulvinar neque. Molestie nunc non blandit massa enim nec dui nunc mattis.
Habitant morbi tristique senectus et. Mattis rhoncus urna neque viverra justo nec. Ultricies integer quis auctor
elit sed vulputate mi sit. Justo eget magna fermentum iaculis.

1.2 Code

```cpp
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <queue>
#include <semaphore.h>
#include <time.h>
using std::queue;

queue<char*> q;
const int max_size = 5;
FILE * file;
int done = 0;

sem_t lock;
sem_t full;
sem_t empty;

void * produce(void * param)
{
    char * text;
    while(!done)
    {
        sem_wait(&empty);
        sem_wait(&lock);
        text = (char*)malloc(255 * sizeof(char));
        if (fgets(text, 256, file) != NULL)
        {
            q.push(text);
            printf("%s %d [!] Text has been added! \n\n", (char*)param, pthread_self());
        }
        else
        {
            done = 1;
        }
        sem_post(&lock);
        sem_post(&full);
        sleep(rand() % 4); // random sleep;
    }
    return NULL;
}

void * consume(void * param)
{
    while(!done || !q.empty()) // when both conditions is False exit the loop;
    {
        sem_wait(&full);
        sem_wait(&lock);
        if(!q.empty())
        {
            printf("%s: %d [!] Text: \n-----------------------------------\n %s
    -------------------------------------\n",(char*)param , pthread_self(), q.front());
            q.pop();
        }
        sem_post(&lock);
        sem_post(&empty);
        sleep(rand() % 4); // random sleep;
    }
    return NULL;
}

int main(int argv, char * argc[])
{
    srand(time(0));

    sem_init(&lock, 0, 1);
    sem_init(&full, 0, 0);
    sem_init(&empty,0, max_size);
    file = fopen("lorem_ipsum.txt", "r");

    pthread_t cons1, cons2;
    pthread_t prod1, prod2;

    pthread_create(&prod1, NULL, produce, (void*)"producer1");
    pthread_create(&prod2, NULL, produce, (void*)"producer2");
    pthread_create(&cons1, NULL, consume, (void*)"consumer1");
    pthread_create(&cons2, NULL, consume, (void*)"consumer2");

    pthread_join(prod1, NULL);
    pthread_join(prod2, NULL);
    pthread_join(cons1, NULL);
    pthread_join(cons2, NULL);

    return 0;
}
```

1.3 Programm output

```
producer1 -2053208320 [!] Text has been added!

consumer2: 2147481344 [!] Text:
-----------------------------------
 Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
-------------------------------------
producer2 -2061601024 [!] Text has been added!

consumer1: -2069993728 [!] Text:
-----------------------------------
 tempor incididunt ut labore et dolore magna aliqua. Nulla facilisi nullam vehicula
-------------------------------------
producer2 -2061601024 [!] Text has been added!

producer2 -2061601024 [!] Text has been added!

producer1 -2053208320 [!] Text has been added!

producer2 -2061601024 [!] Text has been added!
```

```
producer2 -2061601024 [!] Text has been added!

consumer2: 2147481344 [!] Text:
----------------------------------
 ipsum a arcu cursus vitae. Donec et odio pellentesque diam volutpat commodo. In pellentesque massa
-------------------------------------
consumer2: 2147481344 [!] Text:
----------------------------------
 placerat duis ultricies lacus sed. Enim nec dui nunc mattis enim. Id consectetur purus
-------------------------------------
producer1 -2053208320 [!] Text has been added!

consumer1: -2069993728 [!] Text:
----------------------------------
 ut faucibus pulvinar elementum integer. Pharetra pharetra massa massa ultricies.
-------------------------------------
consumer2: 2147481344 [!] Text:
----------------------------------
 Pretium quam vulputate dignissim suspendisse in est ante. Sapien nec sagittis
-------------------------------------
consumer1: -2069993728 [!] Text:
----------------------------------
 aliquam malesuada bibendum. Quis hendrerit dolor magna eget est lorem ipsum. Amet cursus sit amet dict
-------------------------------------
producer2 -2061601024 [!] Text has been added!

consumer1: -2069993728 [!] Text:
----------------------------------
 Leo integer malesuada nunc vel risus commodo. Arcu non odio euismod lacinia.
-------------------------------------
consumer1: -2069993728 [!] Text:
----------------------------------
 Orci ac auctor augue mauris. Eu non diam phasellus vestibulum lorem. Nibh tortor id aliquet lectus pro
-------------------------------------
producer2 -2061601024 [!] Text has been added!

consumer1: -2069993728 [!] Text:
----------------------------------
 Proin fermentum leo vel orci porta non pulvinar neque. Molestie nunc non blandit massa enim nec dui nu
-------------------------------------
producer1 -2053208320 [!] Text has been added!

consumer2: 2147481344 [!] Text:
----------------------------------
 Habitant morbi tristique senectus et. Mattis rhoncus urna neque viverra justo nec. Ultricies integer q
-------------------------------------
producer1 -2053208320 [!] Text has been added!

consumer1: -2069993728 [!] Text:
----------------------------------
 elit sed vulputate mi sit. Justo eget magna fermentum iaculis.
-------------------------------------
```

2 Продовження розминки.

Теж саме, але не на семафорах, а на м'ютексі і умовних змінних. C++

2.1 Input file

Такий самий (Див 1.1)

2.2 Code

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <queue>
#include <semaphore.h>
#include <time.h>
using std::queue;

queue<char*> q;
FILE * file;
int done = 0;

pthread_mutex_t mutex;

void * produce(void * param)
{
    char * text;
    while(!done)
    {
        pthread_mutex_lock(&mutex);
        text = (char*)malloc(255 * sizeof(char));
        if (fgets(text, 256, file) != NULL)
        {
            q.push(text);
            printf("%s %d [!] Text has been added! \n\n", (char*)param, pthread_self());
        }
        else
        {
            done = 1;
        }
        pthread_mutex_unlock(&mutex);
        sleep(rand() % 4); // random sleep;
    }
    return NULL;

}

void * consume(void * param)
{
    while(!done || !q.empty()) // when both conditions is False exit the loop;
    {
        pthread_mutex_lock(&mutex);
        if(!q.empty())
        {
            printf("%s: %d [!] Text: \n-------------------\n %s--------------------\n",(char
    *)param , pthread_self(), q.front());
            q.pop();
        }
        pthread_mutex_unlock(&mutex);
        sleep(rand() % 4); // random sleep;
    }
    return NULL;
}

int main(int argv, char * argc[])
{
    srand(time(0));

    file = fopen("lorem_ipsum.txt", "r");
```

```c

    pthread_t cons1, cons2;
    pthread_t prod1, prod2;
    pthread_mutex_init(&mutex, NULL);

    pthread_create(&prod1, NULL, produce, (void*)"producer1");
    pthread_create(&prod2, NULL, produce, (void*)"producer2");
    pthread_create(&cons1, NULL, consume, (void*)"consumer1");
    pthread_create(&cons2, NULL, consume, (void*)"consumer2");

    pthread_join(prod1, NULL);
    pthread_join(prod2, NULL);
    pthread_join(cons1, NULL);
    pthread_join(cons2, NULL);

    return 0;
}
```

2.3 Programm output

```
producer1 -1394247936 [!] Text has been added!

producer2 -1402640640 [!] Text has been added!

producer2 -1402640640 [!] Text has been added!

consumer1: -1411033344 [!] Text:
-------------------
 Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
--------------------
consumer2: -1419426048 [!] Text:
-------------------
 tempor incididunt ut labore et dolore magna aliqua. Nulla facilisi nullam vehicula
--------------------
consumer1: -1411033344 [!] Text:
-------------------
 ipsum a arcu cursus vitae. Donec et odio pellentesque diam volutpat commodo. In pellentesque massa
--------------------
producer1 -1394247936 [!] Text has been added!

producer2 -1402640640 [!] Text has been added!

consumer1: -1411033344 [!] Text:
-------------------
 placerat duis ultricies lacus sed. Enim nec dui nunc mattis enim. Id consectetur purus
--------------------
consumer2: -1419426048 [!] Text:
-------------------
 ut faucibus pulvinar elementum integer. Pharetra pharetra massa massa ultricies.
--------------------
producer1 -1394247936 [!] Text has been added!

producer2 -1402640640 [!] Text has been added!

consumer1: -1411033344 [!] Text:
-------------------
 Pretium quam vulputate dignissim suspendisse in est ante. Sapien nec sagittis
--------------------
consumer1: -1411033344 [!] Text:
-------------------
 aliquam malesuada bibendum. Quis hendrerit dolor magna eget est lorem ipsum. Amet cursus sit amet dict
--------------------
producer1 -1394247936 [!] Text has been added!
```

```
producer2 -1402640640 [!] Text has been added!

consumer1: -1411033344 [!] Text:
-------------------
 Leo integer malesuada nunc vel risus commodo. Arcu non odio euismod lacinia.
--------------------
consumer1: -1411033344 [!] Text:
-------------------
 Orci ac auctor augue mauris. Eu non diam phasellus vestibulum lorem. Nibh tortor id aliquet lectus pro
--------------------
producer2 -1402640640 [!] Text has been added!

consumer1: -1411033344 [!] Text:
-------------------
 Proin fermentum leo vel orci porta non pulvinar neque. Molestie nunc non blandit massa enim nec dui nu
--------------------
producer1 -1394247936 [!] Text has been added!

consumer2: -1419426048 [!] Text:
-------------------
 Habitant morbi tristique senectus et. Mattis rhoncus urna neque viverra justo nec. Ultricies integer q
--------------------
producer1 -1394247936 [!] Text has been added!

consumer2: -1419426048 [!] Text:
-------------------
 elit sed vulputate mi sit. Justo eget magna fermentum iaculis.
```

3 Продовження розминки для тих, хто шукає пригод. Взаємне блокування

Модифікуйте програму п. 1 так, щоби викликати взаємне блокування. Для цього поміняйте місцями семафори. Переконайтесь у факті взаємного блокування і отримайте задоволення. C++

3.1 Input file

Такий самий (Див 1.1)

3.2 Code

```cpp
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <queue>
#include <semaphore.h>
#include <time.h>
using std::queue;

queue<char*> q;
const int max_size = 5;
FILE * file;
int done = 0;

sem_t lock;
sem_t full;
sem_t empty;

void * produce(void * param)
{
    char * text;
    while(!done)
    {
        sem_wait(&lock);
        sem_wait(&empty);
        text = (char*)malloc(255 * sizeof(char));
        if (fgets(text, 256, file) != NULL)
        {
            q.push(text);
            printf("%s %d [!] Text has been added! \n\n", (char*)param, pthread_self());
        }
        else
        {
            done = 1;
        }
        sem_post(&lock);
        sem_post(&full);
        sleep(rand() % 4); // random sleep;
    }
    return NULL;

}

void * consume(void * param)
{
    while(!done || !q.empty()) // when both conditions is False exit the loop;
    {
        sem_wait(&lock);
        sem_wait(&full);
        if(!q.empty())
        {
            printf("%s: %d [!] Text: \n-----------------------------------\n %s
    -----------------------------------\n",(char*)param , pthread_self(), q.front());
            q.pop();
        }
        sem_post(&lock);
        sem_post(&empty);
        sleep(rand() % 4); // random sleep;
    }
    return NULL;
}

int main(int argv, char * argc[])
{
    srand(time(0));

    sem_init(&lock, 0, 1);
    sem_init(&full, 0, 0);
    sem_init(&empty,0, max_size);
    file = fopen("lorem_ipsum.txt", "r");

    pthread_t cons1, cons2;
    pthread_t prod1, prod2;

    pthread_create(&prod1, NULL, produce, (void*)"producer1");
    pthread_create(&prod2, NULL, produce, (void*)"producer2");
    pthread_create(&cons1, NULL, consume, (void*)"consumer1");
    pthread_create(&cons2, NULL, consume, (void*)"consumer2");

    pthread_join(prod1, NULL);
    pthread_join(prod2, NULL);
    pthread_join(cons1, NULL);
    pthread_join(cons2, NULL);

    return 0;
}
```

3.3 Programm output

```
producer1 457340672 [!] Text has been added!

producer2 448947968 [!] Text has been added!

consumer1: 335542016 [!] Text:
----------------------------------
 Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
-------------------------------------
consumer2: 440555264 [!] Text:
----------------------------------
 tempor incididunt ut labore et dolore magna aliqua. Nulla facilisi nullam vehicula
-------------------------------------

// Programm stucks
```

4 Індивідуальне завдання. Варіант 1а.

Обчислення числа π

4.1 Code

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

int step;

int get_sign(int num)
{
    if (num % 2 == 1)
        return -1;
    else
        return 1;
}

void * get_pi(void * param)
{
    double * sum = (double*)malloc(sizeof(double));
    *sum = 0;

    // 1.000.000.000 - the number of operations made by one thread;
    for(int i = *(int*)param; i < 1000000000; i += step)
    {
        double val = (double)get_sign(i - 1)/(double)(2 * i - 1);
        *sum += val;
    }
    printf("Thread: %d | Sum: %.10f\n", *(int*)param, (*sum) * 4);
    return (void*)sum;
}

int main(int argv, char * argc[])
{
    if (argv != 2)
        return 1;

    const int threads_number = atoi(argc[1]);
    step = threads_number;
    printf("%d\n", threads_number);
    pthread_t threads[threads_number];

    for(int i = 0; i < threads_number; i++)
    {
        int * thread_number = (int*)malloc(sizeof(int));
        *thread_number = i+1;
        pthread_create(&(threads[i]), NULL, get_pi, (void*)(thread_number));
    }
    void * status;
    double sum;
    for(int i = 0; i < threads_number; i++)
    {
        pthread_join(threads[i], &status);
        sum += *((double*)status);
    }
    printf("%.30f\n", sum*4);
}
```

4.2 Programm output

1. Потоки: 4

```
./main_pi_number 4
The number of threads: 4
Thread: 4 | Sum: -10.0704942724
Thread: 1 | Sum: 13.8627320688
Thread: 3 | Sum: 10.3948401205
Thread: 2 | Sum: -11.0454852622
3.141592654591443434242137300316
```

2. Потоків: 10

```
./main_pi_number 10
The number of threads: 10
Thread: 7 | Sum: 3.9582060413
Thread: 1 | Sum: 7.7837051461
Thread: 5 | Sum: 4.1308438777
Thread: 4 | Sum: -4.2783503225
Thread: 2 | Sum: -5.0883348170
Thread: 10 | Sum: -3.8166580709
Thread: 6 | Sum: -4.0313279990
Thread: 3 | Sum: 4.5296268550
Thread: 9 | Sum: 3.8551902692
Thread: 8 | Sum: -3.9013083252
3.141592654590677380355145942303
```

3. Потоків: 21

```
./main_pi_number 21
The number of threads: 21
Thread: 7 | Sum: 0.2597254755
Thread: 8 | Sum: -0.2206794459
Thread: 9 | Sum: 0.1911385221
Thread: 6 | Sum: -0.3135240554
Thread: 1 | Sum: 3.9358035006
Thread: 21 | Sum: 0.0679289448
Thread: 20 | Sum: -0.0720825669
Thread: 4 | Sum: -0.5164410126
Thread: 11 | Sum: 0.1495996492
Thread: 18 | Sum: -0.0819575881
Thread: 16 | Sum: -0.0946316452
Thread: 10 | Sum: -0.1680698085
Thread: 14 | Sum: -0.1114067237
Thread: 13 | Sum: 0.1219713723
Thread: 5 | Sum: 0.3920000009
Thread: 3 | Sum: 0.7422297324
Thread: 2 | Sum: -1.2725068838
Thread: 12 | Sum: -0.1345090456
Thread: 15 | Sum: 0.1023966007
Thread: 19 | Sum: 0.0767292139
Thread: 17 | Sum: 0.0878784180
3.141592654588218014310996295535
```

4. Потоків: 40

```
The number of threads: 40
Thread: 5 | Sum: 1.3164628965
Thread: 1 | Sum: 4.8795613042
Thread: 3 | Sum: 1.6756619928
Thread: 17 | Sum: 0.9753636539
```

```
Thread: 12 | Sum: -1.0348055360
Thread: 6 | Sum: -1.2339195141
Thread: 15 | Sum: 0.9946777186
Thread: 14 | Sum: -1.0062413923
Thread: 16 | Sum: -0.9844655457
Thread: 13 | Sum: 1.0194745976
Thread: 18 | Sum: -0.9671857105
Thread: 20 | Sum: -0.9530464350
Thread: 2 | Sum: -2.2109109482
Thread: 39 | Sum: 0.8837518345
Thread: 10 | Sum: -1.0743718740
Thread: 21 | Sum: 0.9468747594
Thread: 24 | Sum: -0.9310626161
Thread: 7 | Sum: 1.1762933849
Thread: 19 | Sum: 0.9597854539
Thread: 8 | Sum: -1.1336359177
Thread: 29 | Sum: 0.9109739392
Thread: 33 | Sum: 0.8985496355
Thread: 30 | Sum: -0.9076220788
Thread: 31 | Sum: 0.9044440858
Thread: 37 | Sum: 0.8882764306
Thread: 22 | Sum: -0.9411938543
Thread: 40 | Sum: -0.8816176832
Thread: 38 | Sum: -0.8859693537
Thread: 25 | Sum: 0.9265158102
Thread: 28 | Sum: -0.9145173433
Thread: 4 | Sum: -1.4452387943
Thread: 11 | Sum: 1.0528249967
Thread: 9 | Sum: 1.1006790416
Thread: 36 | Sum: -0.8906799226
Thread: 26 | Sum: -0.9222630166
Thread: 35 | Sum: 0.8931874523
Thread: 32 | Sum: -0.9014244786
Thread: 23 | Sum: 0.9359406291
Thread: 27 | Sum: 0.9182725719
Thread: 34 | Sum: -0.8958075197
3.14159265458906622470180991513 2
```

5 Індивідуальне завдання. Варіант 16.

Обчислення π аж поки не набридне

5.1 Code

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <signal.h>

int step;
int done = 0;
pthread_mutex_t mutex;


int get_sign(int num)
{
    if (num % 2 == 1)
        return -1;
    else
        return 1;
}


int next_check = 0;

void * get_pi(void * param)
{
    double * sum = (double*)malloc(sizeof(double));
    *sum = 0;
    int i = *(int*)param;

    while(1)
    {
        pthread_mutex_lock(&mutex);
        next_check += 10000;
        pthread_mutex_unlock(&mutex);
        for(; i < next_check; i += step)
        {
            double val = (double)get_sign(i - 1)/(double)(2 * i - 1);
            *sum += val;
        }
        if(!done)
            continue;
        else
        {
            printf("Thread: %d | Sum: %.10f\n", *(int*)param, (*sum) * 4);
            pthread_exit((void*)sum);
        }
    }

}

void sigint_handler(int sig)
{
    printf("\nGot Signal: %d\n", sig);
    if (sig == 2)
    {
        done = 1;
    }
}

int main(int argv, char * argc[])
{
    if (argv != 2)
        return 1;

    signal(SIGINT, sigint_handler);
    const int threads_number = atoi(argc[1]);
    step = threads_number;
    pthread_t threads[threads_number];
    pthread_mutex_init(&mutex, NULL);
    printf("The number of threads: %d\n", threads_number);

    for(int i = 0; i < threads_number; i++)
    {
        int * thread_number = (int*)malloc(sizeof(int));
        *thread_number = i+1;
        pthread_create(&(threads[i]), NULL, get_pi, (void*)(thread_number));
    }
    void * status;
    double sum;
    for(int i = 0; i < threads_number; i++)
    {
        pthread_join(threads[i], &status);
        sum += *((double*)status);
    }
    printf("Pi number: %.25f\n", sum*4);
}
```

5.2 Program output

1. Приблизно 1 секунда виконання.

```
./main_pi_number_signal 4
The number of threads: 4
^C
Got Signal: 2
Thread: 2 | Sum: -11.0214799434
Thread: 1 | Sum: 13.8387267501
Thread: 3 | Sum: 10.3708348016
Thread: 4 | Sum: -10.0464889576
Pi number: 3.1415926506793532269057323
```

2. Приблизно 2 секунда виконання.

```
./main_pi_number_signal 4
The number of threads: 4
^C
Got Signal: 2
Thread: 4 | Sum: -9.9630677656
Thread: 3 | Sum: 10.2874136140
Thread: 1 | Sum: 13.7553055621
Thread: 2 | Sum: -10.9380587557
Pi number: 3.1415926548321806421881774
```

3. Приблизно 7 секунд виконання

```
./main_pi_number_signal 4
The number of threads: 4
^C
Got Signal: 2
Thread: 2 | Sum: -10.7204736458
Thread: 4 | Sum: -9.7454826608
Thread: 1 | Sum: 13.5377204529
Thread: 3 | Sum: 10.0698285035
Pi number: 3.1415926498115744891492795
```

4. Приблизно 5 секунд виконання з 10 потоками

```
./main_pi_number_signal 10
The number of threads: 10
^C
Got Signal: 2
Thread: 5 | Sum: 3.9751028638
```

```
Thread: 9 | Sum: 3.6994492563
Thread: 3 | Sum: 4.3738858406
Thread: 2 | Sum: -4.9325938024
Thread: 6 | Sum: -3.8755869854
Thread: 1 | Sum: 7.6279641313
Thread: 8 | Sum: -3.7455673121
Thread: 4 | Sum: -4.1226093083
Thread: 7 | Sum: 3.8024650278
Thread: 10 | Sum: -3.6609170558
Pi number: 3.14159265576888380167019 93
```

5. Приблизно 7 секунд виконання з 20 потоками

```
./main_pi_number_signal 20
The number of threads: 20
^C
Got Signal: 2
Thread: 11 | Sum: 1.2634154114
Thread: 1 | Sum: 5.2439507117
Thread: 19 | Sum: 0.1398662615
Thread: 7 | Sum: 0.5243703295
Thread: 5 | Sum: 1.9629374717
Thread: 18 | Sum: -1.6508021123
Thread: 8 | Sum: -0.3686608871
Thread: 20 | Sum: -1.2857671313
Thread: 4 | Sum: -0.9717487678
Thread: 12 | Sum: -1.6687581321
Thread: 9 | Sum: 1.8296835744
Thread: 17 | Sum: 1.5143951665
Thread: 15 | Sum: 1.3400078687
Thread: 10 | Sum: -1.7789068754
Thread: 14 | Sum: -1.7278023938
Thread: 6 | Sum: -1.9663967789
Thread: 2 | Sum: -1.5137887941
Thread: 13 | Sum: 1.7425065355
Thread: 3 | Sum: 2.2159906903
Thread: 16 | Sum: -1.6100748561
Pi number: 3.14159265534212562715811 15
```

Через обмеження double, якщо виконувати довше 10 секунд, процессор встигає зробити стільки операцій, що не вистачає розміру double, і дані починають плисти та число π стає відрізнятися. Щоб вирішити цю проблему, треба використати бібліотеку на java, що дає змогу працювати з великою кількістю знаків після точки.

Також на комп'ютері де я виконую лабораторну роботу встановлений достатньо швидкий процессор Intel Core i7 -3600MQ, тому обчислення відбуваються досить швидко. Меше ніж за секунду мені показує такий результат, `Pi number: 3.14159266288483607354464 77`, що є доволі точним результатом.

6. Якщо зачекати 20 секунд можна отримати такий результат

```
./main_pi_number_signal 4
The number of threads: 4
^C
Got Signal: 2
Thread: 2 | Sum: -2.4247871401
Thread: 3 | Sum: 6.5764089532
Thread: 4 | Sum: -6.5183746496
Thread: 1 | Sum: 11.2669042742
Pi number: 8.90015143773133132754082 91
```

6 Висновки

Впродовж виконання цієї лабораторної роботи я дізнався: як створювати багатопотокові программи, блокувати потоки, коли це потрібно за допомогою семафорів, мютексів. Розглянув такий варіант семафору як locker. Навчився надсилати до программи сигнал SIGINT та обробляти його так, як треба розробнику, без термінового завершення программи.