

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**  
**KHOA ĐIỆN TỬ - VIỄN THÔNG**



**THỰC TẬP LẬP TRÌNH NHÚNG VHT**  
**Dự án gNode 5G**

**Người thực hiện:**

Ngô Sỹ Trung

19020644

QH-2019-I/CQ-R

Khoa Điện tử Viễn thông

**Thời gian thực hiện:**

Từ 07/07/2022 tới 19/7/2022

Hà Nội, 2021

## 1. Nội dung cần thực hiện

Nhiệm vụ được giao là viết chương trình bằng ngôn ngữ lập trình C có nội dung được trình bày như Hình 1.1.

1. Viết chương trình C trên Linux chạy 3 thread **SAMPLE**, **LOGGING**, **INPUT**. Trong đó:
  - Thread **SAMPLE** thực hiện vô hạn lần nhiệm vụ sau với chu kỳ **X** ns. Nhiệm vụ là đọc thời gian hệ thống hiện tại (chính xác đến đơn vị ns) vào biến **T**.
  - Thread **INPUT** kiểm tra file “freq.txt” để xác định chu kỳ **X** (của thread **SAMPLE**) có bị thay đổi không?, nếu có thay đổi thì cập nhật lại chu kỳ **X**. Người dùng có thể echo giá trị chu kỳ **X** mong muốn vào file “freq.txt” để thread **INPUT** cập nhật lại **X**.
  - Thread **LOGGING** chờ khi biến **T** được cập nhật mới, thì ghi giá trị biến **T** và giá trị **interval** (offset giữa biến **T** hiện tại và biến **T** của lần ghi trước) ra file có tên “time\_and\_interval.txt”.
2. Viết shell script để thay đổi lại giá trị chu kỳ **X** trong file “freq.txt” sau mỗi 1 phút. Các giá trị **X** lần lượt được ghi như sau: 1000000 ns, 100000 ns, 10000 ns, 1000 ns, 100ns.
3. Chạy shell script + chương trình C trong vòng 5 phút, sau đó dừng chương trình C.
4. Thực hiện khảo sát file “time\_and\_interval.txt”: Vẽ đồ thị giá trị **interval** đối với mỗi giá trị chu kỳ **X** và đánh giá.

Hình 1.1 Nội dung nhiệm vụ được giao

Yêu cầu chính:

- Làm quen với việc lập trình phân luồng xử lý.
- Tìm hiểu công việc lập trình yêu cầu tính toán trong thời gian thực.
- Học cách đánh giá kết quả thu về, từ đó đưa ra phương thức tối ưu.
- Làm quen với việc lập trình xử dụng dòng lệnh hệ thống (Shell Script)

## 2. Lý thuyết cần tìm hiểu

### 2.1. Phân luồng trong C

Thông thường, việc chỉ sử dụng một quy trình (process) để chạy nhiều tác vụ đem lại hiệu suất không cao do các lệnh phải xử lý lần lượt. Vậy nên để có thể thực hiện nhiều nhiệm vụ đồng thời trong một quy trình cần sử dụng tới chức năng phân luồng. Công việc phân luồng này có mục đích tạo ra các luồng xử lý riêng biệt, tuy nhiên vẫn sử dụng bộ nhớ toàn cục chung và vẫn có thể tác động lẫn nhau.

Ngôn ngữ C cung cấp chức năng tạo ra các luồng xử lý có tên là **pthread** bao gồm các hàm chính như sau.

Hàm khởi tạo luồng:

```
#include <pthread.h>
```

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,  
void *(*start)(void *), void *arg);
```

Returns 0 on success, or a positive error number on error

Hàm thoát luồng:

```
include <pthread.h>
```

```
int pthread_join(pthread_t thread, void **retval);
```

Returns 0 on success, or a positive error number on error

## 2.2. Đồng bộ các luồng trong chương trình

Để đồng bộ hoá các luồng, tránh gây xung đột khi nhiều luồng tác động vào cùng một biến toàn cục, pthread cung cấp hàm khởi tạo mutex phục vụ cho việc đồng bộ dữ liệu giữa các luồng.

Hàm khởi tạo mutex như sau:

```
pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
```

Sau khi khởi tạo các lệnh cần đồng bộ sẽ được thực hiện trong các khoá mutex, hàm khoá và mở khoá mutex như sau:

```
#include <pthread.h>
```

```
int pthread_mutex_lock(pthread_mutex_t *mutex);  
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Both return 0 on success, or a positive error number on error

## 2.3. Hàm tạo thời gian nghỉ (sleep)

Trong C có nhiều hàm tạo thời gian nghỉ, trong đó có thể kể đến như sleep(), usleep(), nếu cần độ phân giải tốt hơn có thể dùng tới hàm nanosleep() và clock\_nanosleep().

Trong đó, hàm clock\_nanosleep() được sử dụng rộng rãi hơn khi làm việc với chức năng phân luồng bởi hàm này tính toán thời gian nghỉ dựa vào gốc thời gian được lấy ban đầu, điều này giúp giảm ảnh hưởng của việc xử lý các luồng khác tới thời gian nghỉ. Trong khi đó hàm nanosleep() khởi tạo thời gian nghỉ chỉ khi chương trình chạy tới hàm, việc phải xử lý nhiều lệnh trước khi tới hàm nanosleep() có thể làm thời gian ghi nhận có sai số lớn.

Hàm clock\_nanosleep có thể được khởi tạo như sau:

```
int clock_nanosleep(clockid_t clockid, int flags,  
    const struct timespec *request, struct timespec *remain);
```

Returns 0 on successfully completed sleep,  
or a positive error number on error or interrupted sleep

## 2.4. Tập lệnh Shell Scripts

Shell là một chương trình thông dịch lệnh của một hệ điều hành, cung cấp cho người dùng khả năng tương tác với hệ điều hành bằng cách gõ từng lệnh trong terminal, đồng thời trả lại kết quả thực hiện lệnh lại cho người sử dụng.

Shell Script là một tập hợp các lệnh Shell giúp người lập trình giảm thiểu thời gian và tránh sai sót khi phải nhập nhiều lệnh Shell. Shell Script thường bắt đầu bằng dòng lệnh `#!/bin/bash` và lưu dưới định dạng file `.sh`. Cú pháp trong Shell Script tương tự như cú pháp khi gõ trên Terminal vì vậy khá dễ dàng để thực hiện.

### 3. Các chương trình triển khai

Toàn bộ các chương trình chạy và có tệp liên quan được đẩy lên Github tại đường dẫn: [Thread and Time project](#)

Các phần chính trong chương trình được trình bày như sau.

#### 3.1. Chương trình C

Chương trình C chạy ba luồng chính như đề bài đặt ra được trình bày chi tiết dưới đây.

##### 3.1.1. Luồng INPUT

Luồng INPUT có chức năng nhận giá trị X mong muốn từ tệp “freq.txt” để làm tham chiếu cho chu kỳ lấy mẫu. Nếu giá trị X trong tệp thay đổi thì cập nhật lại giá trị X.

```
void *Input_function(void *INPUT)
{
    while(1){
        // Read input from freq.txt
        fp_X = fopen("freq.txt", "r");
        fscanf (fp_X, "%d", &X);

        // Notify Sample thread about X frequency
        pthread_mutex_lock(&mtx_input);
        input_flag = 1;
        pthread_cond_signal(&condition_input);
        pthread_mutex_unlock(&mtx_input);

        // printf("%d \n", X);
        fclose(fp_X);
    }
}
```

##### 3.1.2. Luồng SAMPLE

Luồng SAMPLE ghi nhận thời gian hiện tại của hệ thống theo chu kỳ X nhận được từ luồng INPUT, thời gian được lưu vào biến T để luồng LOGGING tiếp tục xử lý.

```
void *Sample_function(void * SAMPLE)
{
    // Read current date and time of the system
    clock_gettime(CLOCK_REALTIME, &ts);
    char buff[100];

    while(1)
    {
```

```

        // check if X frequency changed (Input)
        pthread_mutex_lock(&mtx_input);
        while(input_flag ==0)
        {
            pthread_cond_wait(&condition_input, &mtx_input);
        }
        pthread_mutex_unlock(&mtx_input);

        ts.tv_nsec += X;

        if(ts.tv_nsec > 1000000000)
        {
            // temp = ts.tv_nsec;
            ts.tv_nsec = ts.tv_nsec - 1000000000;
            ts.tv_sec++;
        }
        clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME,
&ts, NULL);

        // Save current time
        clock_gettime(CLOCK_REALTIME, &current);

        // ts.tv_nsec += X;
        // // Sleep X nanosecond
        // clock_nanosleep(CLOCK_REALTIME,    TIMER_ABSTIME,
&ts, NULL);
        // clock_gettime(CLOCK_REALTIME, &current);
        snprintf(T, sizeof T, "%ld.%09ld offset: %ld.%09ld
\n", current.tv_sec, current.tv_nsec, diff_sec, diff_nsec);

        // return flag for Input
        input_flag = 0;

        // Create signal for LOGGING thread
        pthread_mutex_lock(&mtx_sample);
        sample_flag = 1;
        pthread_cond_signal(&condition_sample);
        pthread_mutex_unlock(&mtx_sample);
    }
}

```

### 3.1.3. Luồng LOGGING

Khi có giá trị T nhận được từ luồng SAMPLE, luồng LOGGING tiến hành tính toán sai lệch so với thời gian ngay trước đó, sau đó ghi giá trị thời gian và sai số vào tệp “time\_and\_interval.txt”.

Ngoài ra luồng LOGGING còn ghi giá trị sai số ra tệp “.txt” tương ứng với giá trị chu kỳ X tham chiếu để thuận tiện hơn cho việc vẽ đồ thị.

```
void *Logging_function(void *LOGGING){

    while(1){
        pthread_mutex_lock(&mtx_sample);
        while(sample_flag ==0)
        {
            pthread_cond_wait(&condition_sample,
&mtx_sample);
        }

        // Point to offset_data_X.txt
        fp_offset = fopen(file_name,"a");

        // Calculate offset
        // offset = ((double)ts.tv_sec + 1.0e-9*ts.tv_nsec) -
        ((double)ts_previous.tv_sec + 1.0e-9*ts_previous.tv_nsec);
        // printf("%.9f seconds\n", offset);

        diff_sec      =      ((long)      current.tv_sec)      -
ts_previous.tv_sec ;
        diff_nsec;

        if(ts_previous.tv_nsec      !=      current.tv_nsec      ||
ts_previous.tv_sec != current.tv_sec)
        {
            if(current.tv_nsec > ts_previous.tv_nsec)
            {
                diff_nsec      =      current.tv_nsec      -
ts_previous.tv_nsec;
            }
            else
            {
                diff_nsec = 1000000000 + current.tv_nsec -
ts_previous.tv_nsec ;
                diff_sec = diff_sec - 1;
            }
            fprintf(file,"\n%ld.%09ld", diff_sec, diff_nsec);
        }

        ts_previous.tv_nsec = current.tv_nsec;
        ts_previous.tv_sec  = current.tv_sec;

        // Save offset to specific file name
        // fp_offset = fopen(file_name,"a");
        if (fp_offset){
```

```

        fprintf(fp_offset, "%ld.%09ld\n", diff_sec,
diff_nsec);
    }
    else{
        printf("Failed to open the file \n");
    }
    fclose(fp_offset);

    // Save date and time to "time_and_interval.txt"
    fp = fopen("time_and_interval.txt","a");
    if (fp){
        fputs(T,fp);
    }
    else{
        printf("Failed to open the file \n");
    }
    fclose(fp);

    // Return sample_flag value to continue save time
value
    sample_flag = 0;

    pthread_mutex_unlock(&mtx_sample);

}
}

```

### 3.2. Tập lệnh Shell Script

Tập lệnh Shell Script có chức năng thay đổi giá trị chu kỳ X sau mỗi 1 phút, đồng thời biên dịch và chạy chương trình C trong vòng 5 phút và dừng chương trình.

Tập lệnh được viết chi tiết như sau.

```

#!/bin/bash

# clear or remove data form file
> time_and_interval.txt
rm *offset_data_*

# Compile C program
gcc -pthread -o Thread Time_Interval_Thread.c

# get time with X period for (interval) seconds, can be
changed for better surveying
interval=60

#run 5 times
for i in {6..2}

```



```
do
    X_fake=$((10 ** $i))

    # > offset_data_$(X_fake).txt

    echo "$X_fake" > freq.txt
    echo "X frequency: $X_fake" >> time_and_interval.txt
    echo      "X      frequency:      $X_fake"      >>
offset_data_$(X_fake).txt

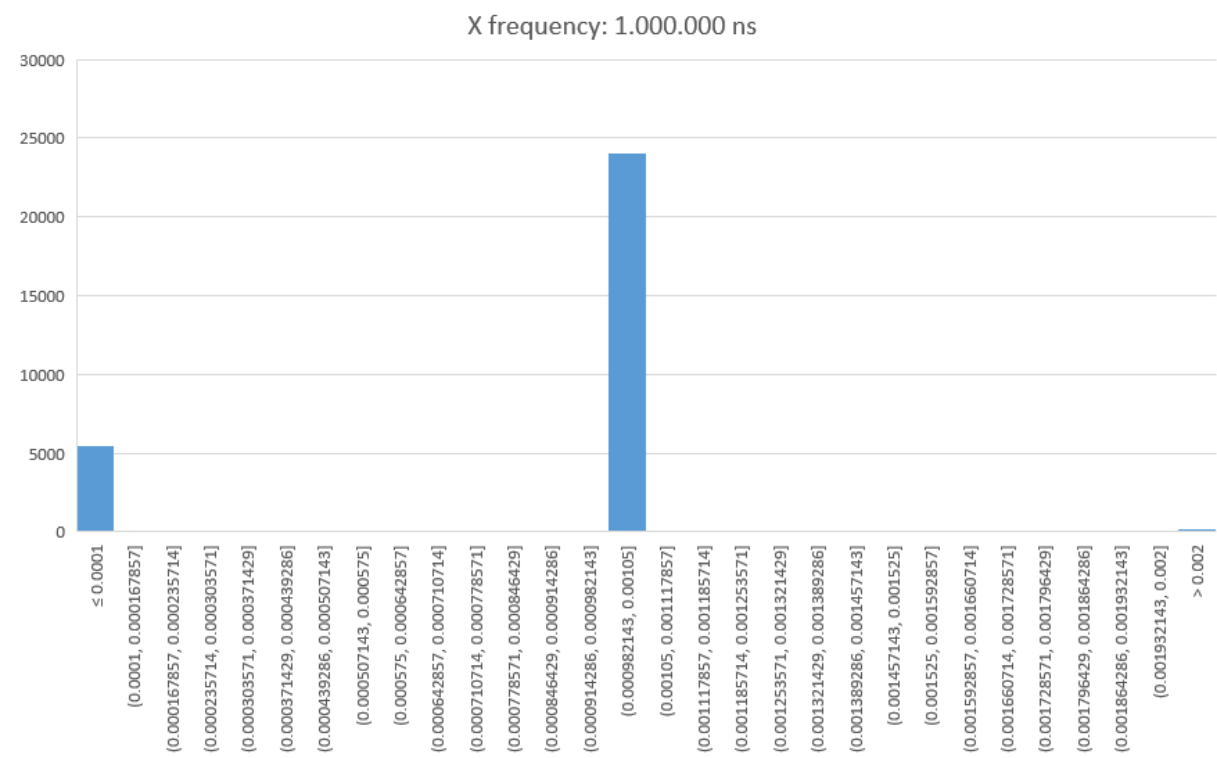
    # run program for (interval) seconds
    timeout $interval ./Thread "$X_fake"
    # sleep 10
done
```

## 4. Kết quả và thảo luận

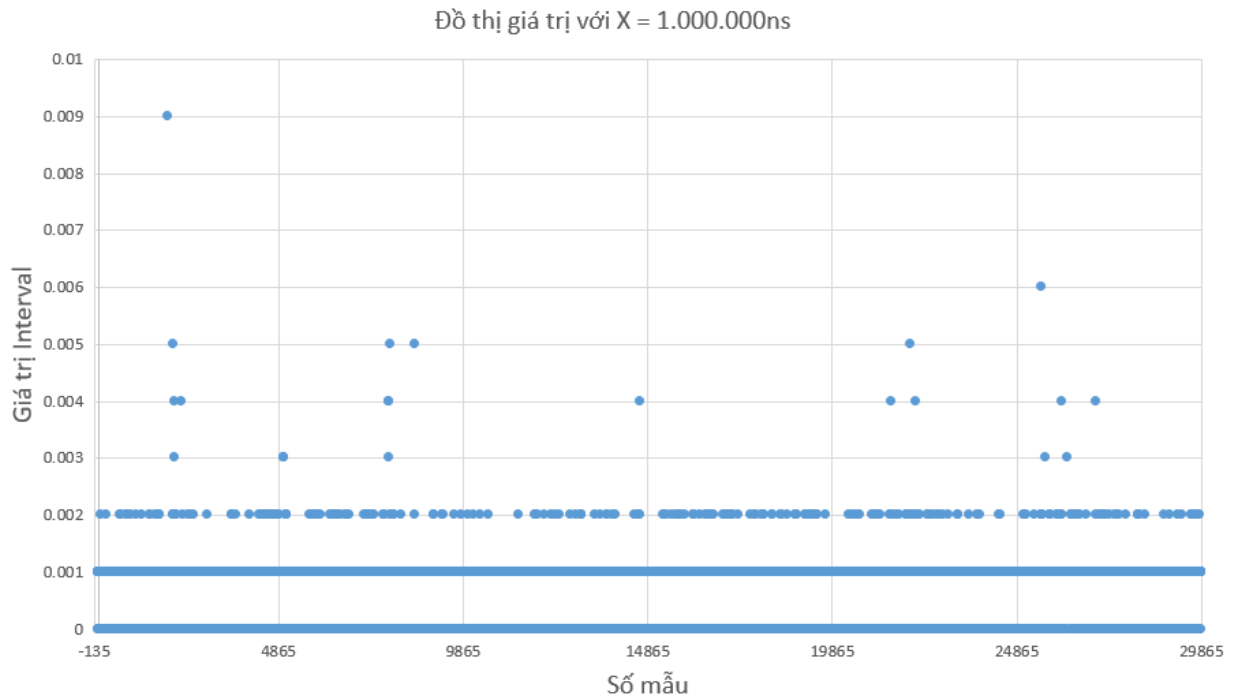
Các mục dưới đây khảo sát kết quả chu kỳ chương trình thu được. Những kết quả này được vẽ đồ thị phân bố giá trị để so sánh với giá trị chu kỳ X mong muốn, từ đó đưa ra đánh giá và kết luận

### 4.1. Khảo sát chu kỳ 1.000.000ns

Đồ thị phân bố và đồ thị giá trị tương ứng với chu kỳ 1.000.000ns được thể hiện như hình Hình 4.1 và Hình 4.2.



Hình 4.1 Đồ thị phân bố ứng với giá trị chu kỳ 1.000.000ns



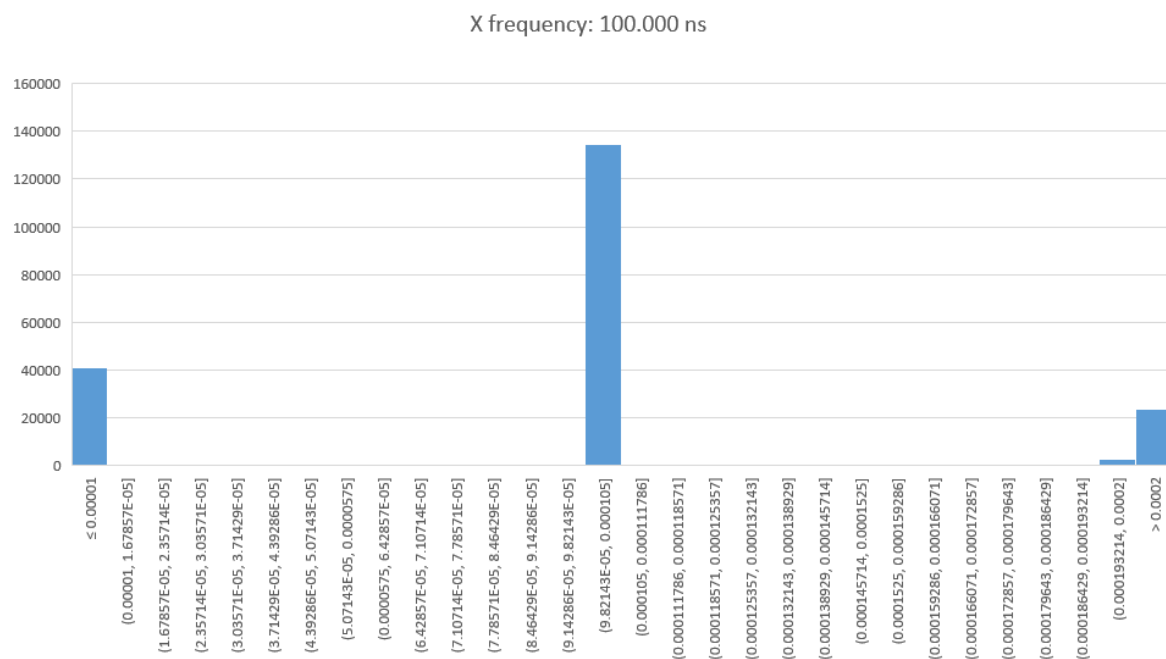
Hình 4.2 Đồ thị giá trị ứng chu kỳ 1.000.000 ns

Lấy mẫu 29.865 mẫu, nhận thấy khoảng 24.500 giá trị (82%) nằm xung quanh giá trị mong muốn: từ  $[0.000982143; 0.00105]$ , chứng tỏ chương trình đã xử lý khá tốt khoản giá trị này.

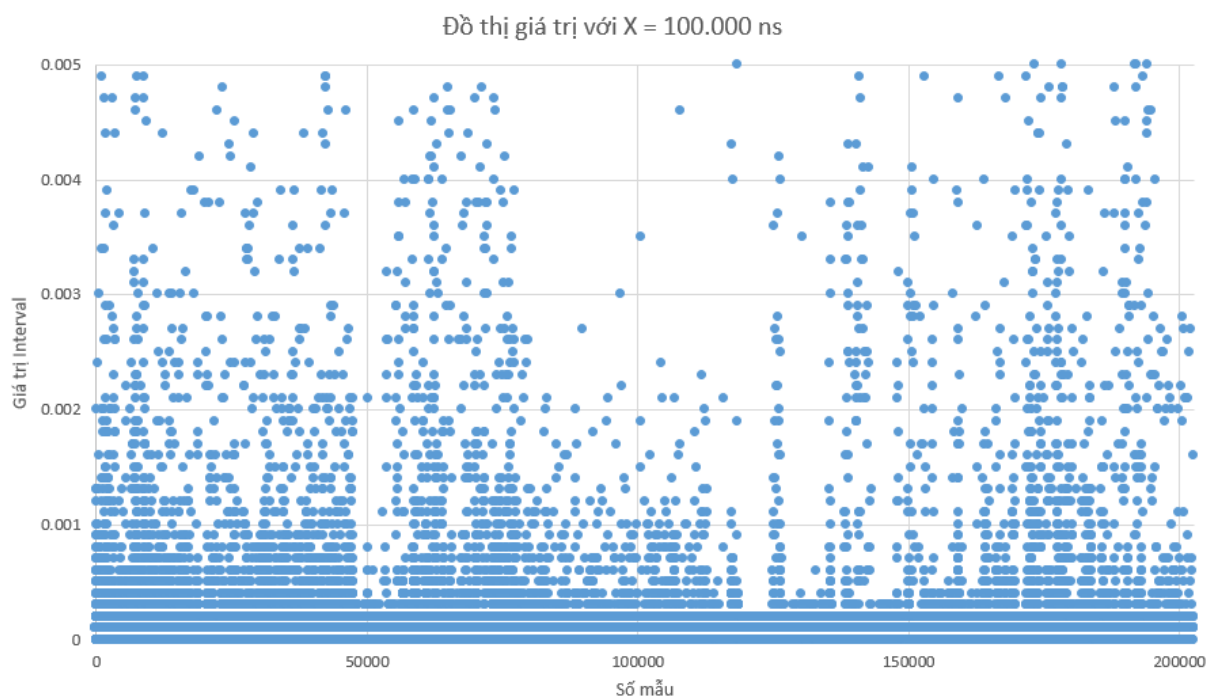
Tuy nhiên rất nhiều mẫu trả về giá trị bằng 0, chứng tỏ chương trình vẫn còn lỗi cần khắc phục.

## 4.2. Khảo sát chu kỳ 100.000ns

Hình 4.3 và Hình 4.4 thể hiện biểu đồ phân bố và đồ thị giá trị khi chu kỳ mong muốn là 100.000 ns



Hình 4.3 Đồ thị phân bố ứng với giá trị 100.000 ns



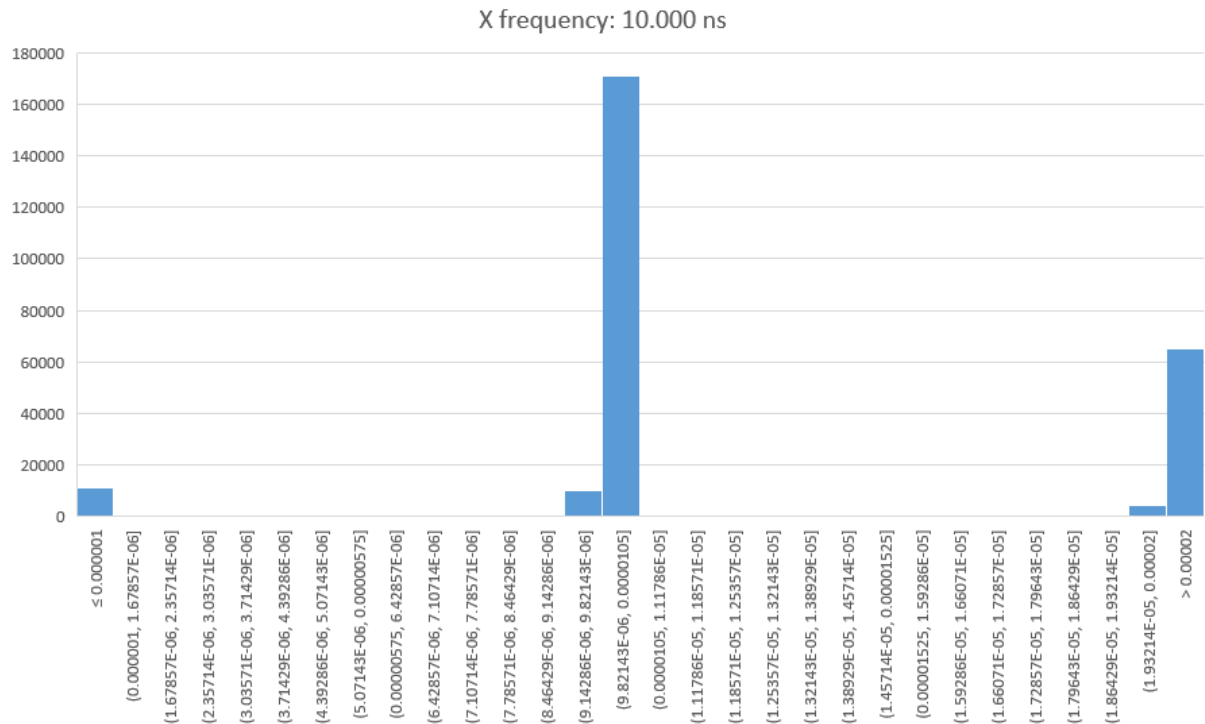
Hình 4.4 Đồ thị giá trị ứng với chu kỳ 100.000 ns

Lấy mẫu khoảng 200000 mẫu, tương tự như trên, 70% giá trị (khoảng 135000) nằm xung quanh giá trị mong muốn  $[0.00009; 0.000105]$  số lượng mẫu mong muốn đã giảm 10% so với chu kỳ 1.000.000ns.

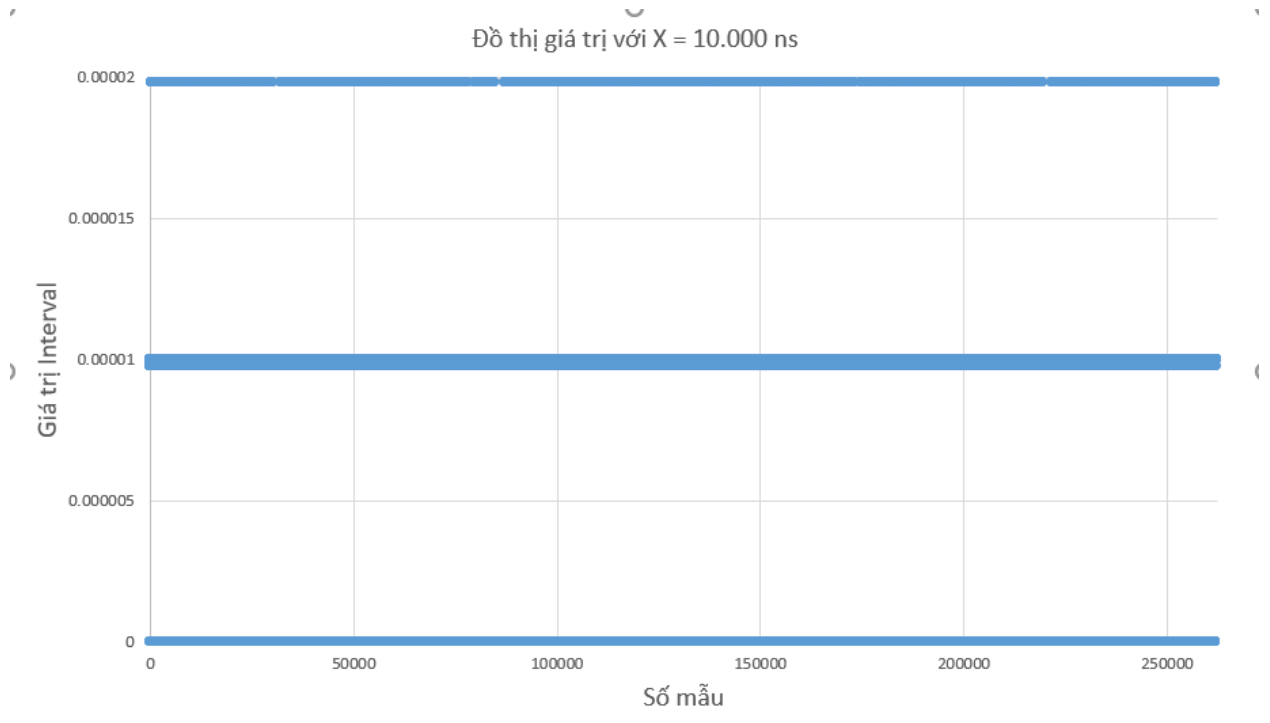
Tương tự như trên, vẫn có khá mẫu với giá trị bằng 0 tương tự trường hợp 1.000.000ns.

### 4.3. Khảo sát chu kỳ 10.000ns

Hình 4.5 và Hình 4.6 biểu diễn giá trị thu được khi lấy mẫu với giá trị 10.000ns



Hình 4.5 Biểu đồ phân bố ứng với chu kỳ 10.000 ns

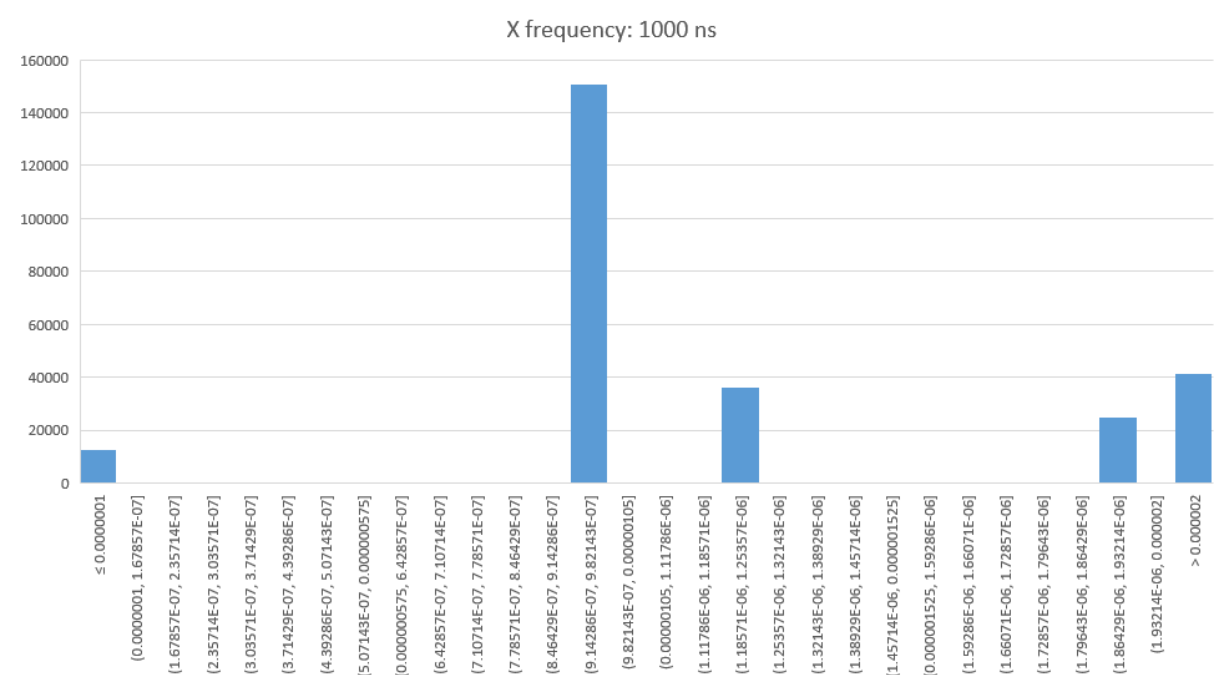


Hình 4.6 Đồ thị giá trị tương ứng với chu kỳ 10.000 ns

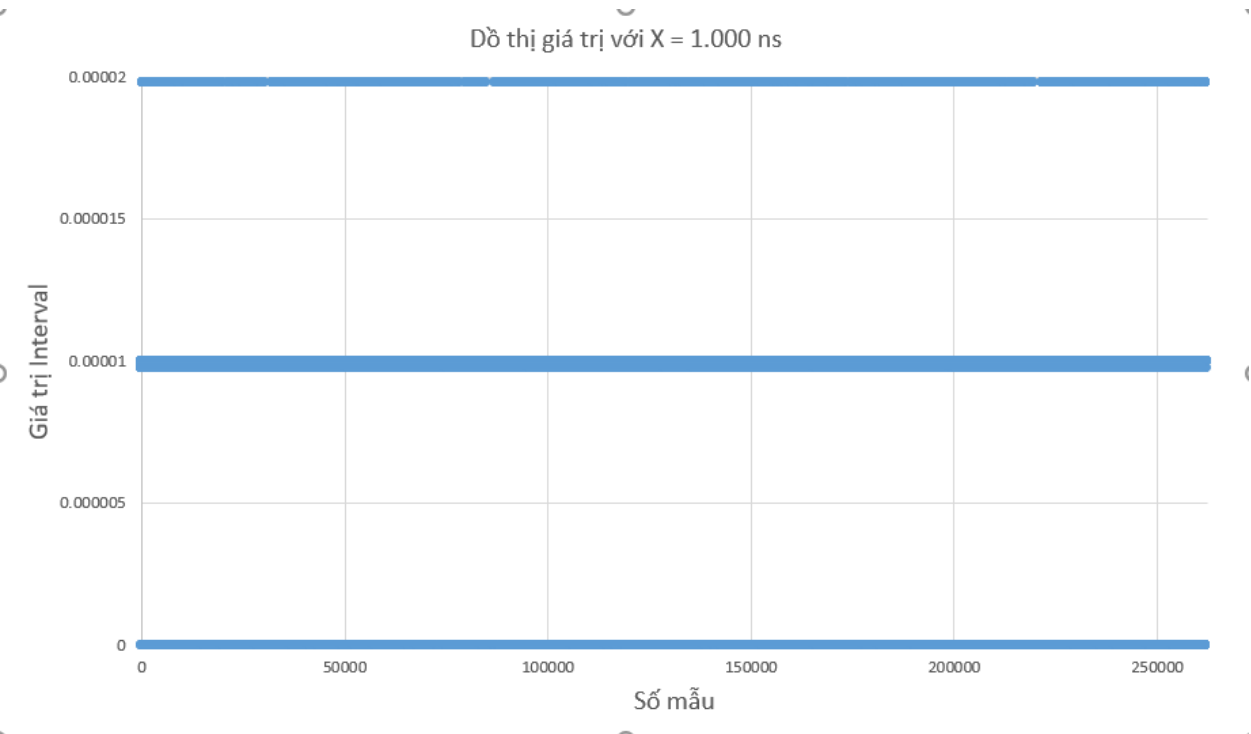
Lấy mẫu khoảng 260000 giá trị, khoảng 70% giá trị nằm xung quanh chu kỳ mong muốn, 23% giá trị không thể đo được dưới chu kỳ 20.000ns, chương trình đã bắt đầu thể hiện tình trạng không thể lấy mẫu mong muốn.

4.4. Khảo sát chu kỳ 1.000ns

Hình 4.7 và Hình 4.8 thể hiện biểu đồ phân bố và đồ thị giá trị ứng với chu kỳ 1.000 ns



Hình 4.7 Biểu đồ phân bố ứng với giá trị chu kỳ 1000 ns

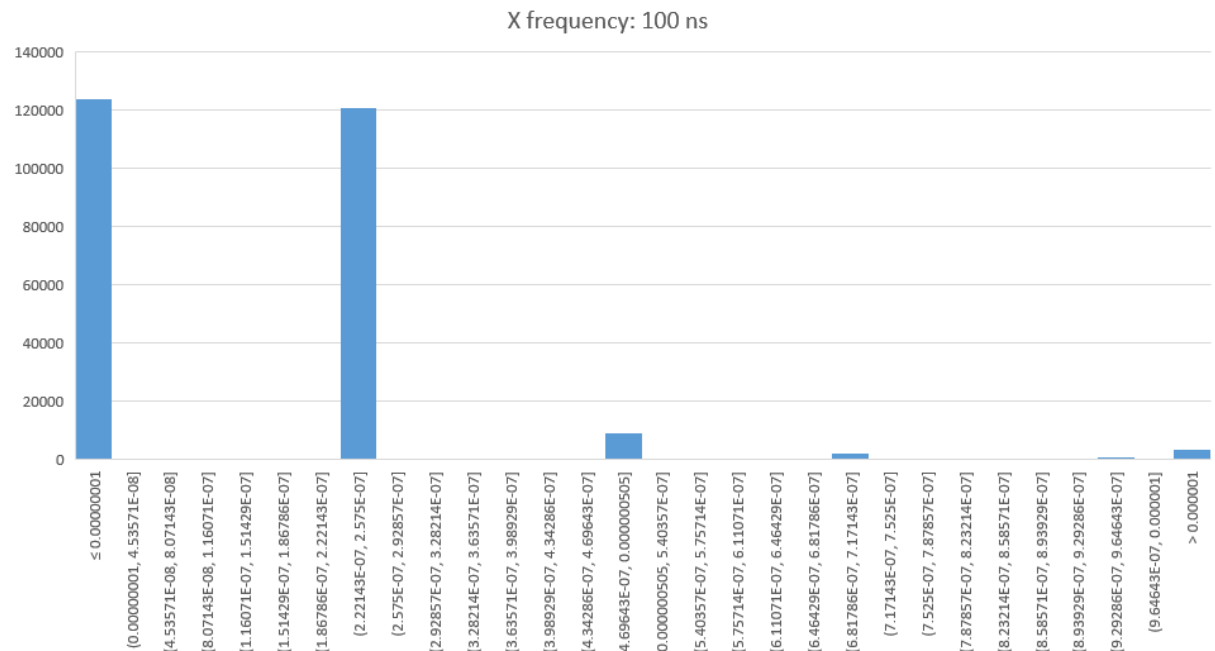


Hình 4.8 Đồ thị giá trị tương ứng với chu kỳ 1000 ns

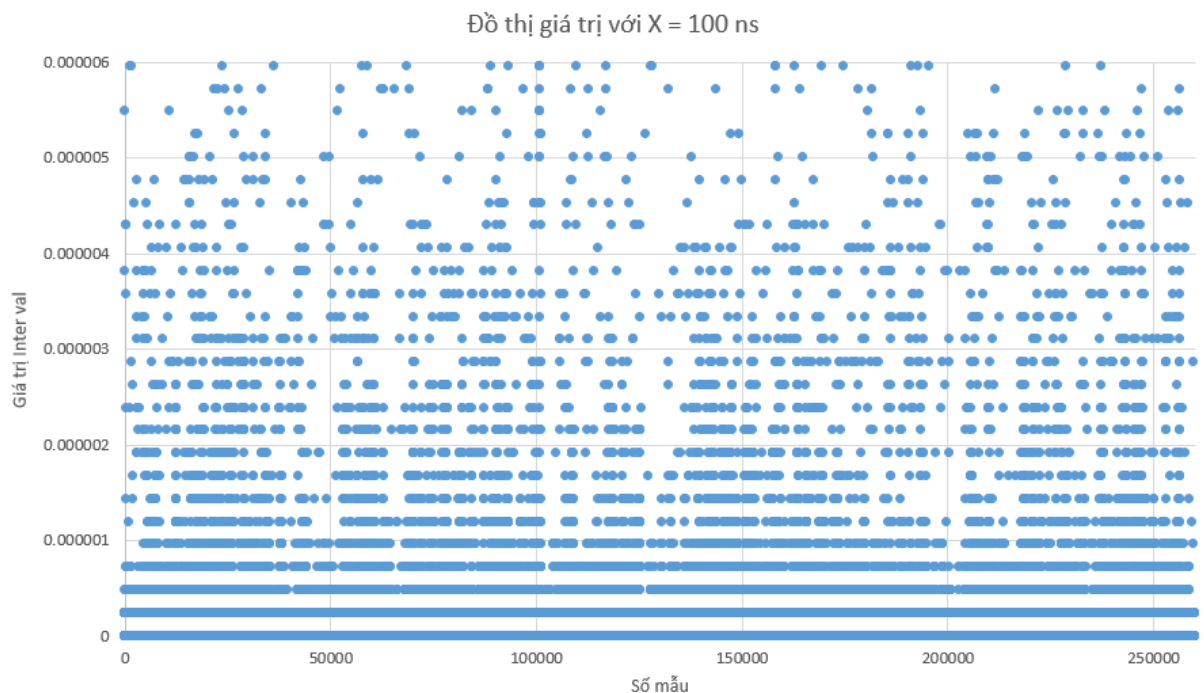
Lấy mẫu được khoảng 270000 giá trị, chỉ còn khoảng 56% giá trị được như mong muốn, 15% giá trị với sai số chấp nhận được. Kết quả trả về tương tự như khi khảo sát với chu kỳ 100.000 ns và 10.000 ns.

## 4.5. Khảo sát chu kỳ 100ns

Với chu kỳ tham chiếu nhỏ nhất, biểu đồ phân bố và đồ thị được thể hiện trong Hình 4.9 và Hình 4.10



Hình 4.9 Biểu đồ phân bố ứng với chu kỳ 100 ns



Hình 4.10 Đồ thị giá trị tương ứng với chu kỳ 100 ns

Số giá trị lấy mẫu được khoảng 270000 mẫu, khoảng 45% giá trị nhỏ hơn 100ns, chứng tỏ chương trình không thể lấy mẫu được, 45% giá trị còn lại thuộc khoảng 222ns tới 257ns. Hệ thống vẫn có thể lấy mẫu với giá trị rất nhỏ được, tuy nhiên thông số cho thấy số lượng cũng không đáng tin cậy.

Nhận xét:

- Chu kỳ càng cao khi giá trị trả về càng chính xác
- Kết quả thu được đối với các trường hợp 1.000.000 ns, 100.000 ns, 10.000 ns và 1.000 ns khá tương đồng với nhau.
- Chương trình có thể lấy được tương đối nhiều mẫu gần với trường hợp chu kỳ nhỏ nhất 100ns.
- Chương trình vẫn gặp phải vấn đề trả về rất nhiều giá trị 0, cần phải tiếp tục sửa lỗi và tối ưu.



## 5. Kết luận

Việc xử lý dữ liệu trong thời gian thực gặp rất nhiều khó khăn khi làm việc với các tham số rất nhỏ, cần phải đọc thêm để nắm rõ cách thức vận hành và quy trình hoạt động của hệ thống mới có thể tiếp tục tối ưu chương trình.

Các vấn đề gặp phải bao gồm:

- Chưa hiểu rõ về Thread và Timer
- Còn hạn chế trong khâu xử lý dữ liệu
- Thuật toán chưa tối ưu