BÁO CÁO

LINUX PROGRAMMING ASSIGMENT

Nhóm thực tập Nhúng – dự án gNodeB 5G VHT

Sinh viên: Nguyễn Mạnh Hùng

MSSV: 18020600

Hà Nội, ngày 19 tháng 7 năm 2022

MỤC LỤC

1.Phân tích yêu cầu bài toán:

Viết chương trình C chạy các luồng(thread). Viết thread SAMPLE để đọc thời gian hiện tại của hệ thống sau 1 khoảng chu kì X(ns). Trong file "freq.txt" có chưa dữ liệu là chu kì X. Viết thread INPUT để kiểm tra file "freq.txt" xem X có thay đổi không, nếu X thay đổi thì cập nhật lại X mới. Viết thread LOGGING ghi giá trị interval (khoảng cách thời gian giữa 2 lần lấy mẫu) và thời gian hiện tại của hệ thống vào file "time_and_interval.txt".

2.Lý thuyết đã tìm hiểu được:

2.1 Multi thread

Thread là một luồng chuỗi đơn trong một quy trình(process). Bởi vì các luồng có một số thuộc tính của các quy trình, chúng đôi khi được gọi là các quy trình nhẹ(lightweight processes).

Các thread không độc lập với nhau không giống như các process. Các luồng chia sẻ với các luồng khác phần mã, phần dữ liệu và tài nguyên hệ điều hành của chúng như các tệp và tín hiệu đang mở. Tuy nhiên, giống như các process, một luồng có bộ đếm chương trình riêng (PC), một bộ thanh ghi và một không gian ngăn xếp.

Multi thread cải thiện hiệu suất, tốc độ thông qua việc chạy song song nhiều thread.

Thread chạy nhanh hơn process vì:

- Tạo thread nhanh hơn process
- Chuyển đổi ngữ cảnh giữa các luồng nhanh hơn
- Thread có thể kết thúc dễ dàng hơn
- Giao tiếp giữa các thread dễ dàng hơn

2.2 Clock nanosleep

Dùng clock nanosleep thay cho nanosleep để tạo thời gian ngủ trong C vì:

- clock_nanosleep thì nó cộng dồn thời gian ngủ từ mốc đã khởi tạo ban đầu, điều này giảm sự liên quan đến chu trình xử lý lệnh của hệ thống. Còn nanosleep thì ngủ khi chu trình code chạy đến lệnh đó, việc tốn thời gian xử lý các lệnh trước nanosleep làm sai lệch thời gian lấy mẫu
- Có thể chỉ định thời gian tuyệt đối để ngủ cho đến thay vì một khoảng thời gian để ngủ. Điều này tạo ra sự khác biệt cho đồng hồ thời gian thực (thời gian treo tường), có thể được đặt lại bởi quản trị viên. Với nanosleep cần tính toán trước khoảng thời gian ngủ để đạt đến một thời gian tuyệt đối nhất định. Ta sẽ không thể thức dậy nếu đồng hồ đặt lại và thời gian mong muốn đến "sớm". Ngoài ra, nếu tính toán khoảng thời gian muốn ngủ, nhưng lại được gọi trước khi gọi nanosleep và không được lên lịch lại trong một thời gian, hệ thống sẽ lại ngủ quá lâu.

2.3 Thread synchronization

Thread synchronization được định nghĩa là một cơ chế đảm bảo rằng hai hoặc nhiều quy trình hoặc luồng đồng thời không thực hiện đồng thời một số phân đoạn chương trình cụ thể được gọi là phần quan trọng. Quyền truy cập của các quy trình vào phần quan trọng được kiểm soát bằng cách sử dụng các kỹ thuật đồng bộ hóa. Khi một luồng bắt đầu thực hiện phần quan trọng (một phân đoạn được tuần tự hóa của chương trình), luồng kia sẽ đợi cho đến khi luồng đầu tiên kết thúc. Nếu các kỹ thuật đồng bộ hóa thích hợp không được áp dụng, nó có thể gây ra tình trạng chạy đua trong đó giá trị của các biến có thể không thể đoán trước và thay đổi tùy thuộc vào thời gian chuyển mạch ngữ cảnh của các quy trình hoặc luồng.

Khi 1 biến bị nhiều thread truy cập vào cùng 1 lúc sẽ làm sao chương trình chạy sai. Cách khắc phục là sử dụng mutex.

Làm việc của một mutex:

- Giả sử một luồng đã khóa một vùng mã bằng mutex và đang thực thi đoạn mã đó.
- Bây giờ nếu bộ lập lịch quyết định thực hiện chuyển đổi ngữ cảnh, thì tất cả các luồng khác đã sẵn sàng thực thi cùng một vùng sẽ được bỏ chặn.

- Chỉ một trong số tất cả các luồng sẽ thực hiện nó nhưng nếu luồng này cố gắng thực thi cùng một vùng mã đã bị khóa thì nó sẽ lại chuyển sang trạng thái ngủ.
- Quá trình chuyển đổi ngữ cảnh sẽ diễn ra lặp đi lặp lại nhưng không luồng nào có thể thực thi vùng mã bị khóa cho đến khi khóa mutex trên nó được giải phóng.
- Khóa Mutex sẽ chỉ được phát hành bởi chủ đề đã khóa nó.
- Vì vậy, điều này đảm bảo rằng khi một luồng đã khóa một đoạn mã thì không một luồng nào khác có thể thực thi cùng một vùng cho đến khi nó được mở khóa bởi luồng đã khóa nó.

2.4 Shell script

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 ở chế độ dòng lệnh, đồng thời trả lại kết quả thực hiện lệnh lại cho người sử dụng. Shell cung cấp tập hợp các lệnh đặc biệt mà từ đó có thể tạo nên chương trình, khi đó được gọi là shell script.

Về cơ bản shell script là 1 tập hợp các lệnh được thực thi nối tiếp nhau

Trước khi làm bất cứ điều gì với script, cần thông báo với system rằng chuẩn bị có shell chạy bằng dòng lệnh #!/bin/bash. Để soạn thảo shell script thì có thể gõ ngay trên terminal hoặc dùng các trình soạn thảo nào như vim, gedit, kate,... sau đó lưu lại file *.sh, ví dụ test.sh. Sau đó thiết lập quyền thực thi cho shell $chmod +x test.sh. Rồi chạy script bằng 1 trong 3 cách: bash test.sh, sh test.sh, ./test.sh

3.Triển khai

3.1 Thread SAMPLE

Thread SAMPLE thực hiện vô hạn lần nhiệm vụ sau với chu kì X ns. Nhiệm vụ là đọc thời gian hệ thống hiện tại (chính xác đến đơn vị ns) vào biến T.

```
struct timespec T;
struct timespec tp;
struct timespec tmp;
long freq;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void sleep_until(struct timespec *ts, int delay)
{
	ts->tv_nsec += delay;
	if(ts->tv_nsec > 1000*1000*1000){
		ts->tv_nsec -= 1000*1000*1000;
		ts->tv_sec++;
	}
	clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, ts,  NULL);
}

long get_freq()
{
	long data;
	FILE *f;
	f = fopen("freq.txt","r");
	char buff[100];
	fgets(buff,sizeof(buff),f);
	char *eptr;
	data = strtol(buff,&eptr,10);
	fclose(f);
	return data;
```

```
 }

void *getTime(void *args)
{       long freq = *((long*)args);
        clock_gettime(CLOCK_REALTIME,&tmp);
        while (1)
        {

                sleep_until(&tmp, freq);
                //save thoi gian thuc vao bien T
                clock_gettime(CLOCK_REALTIME,&T);

        }

}
```

3.2 Thread INPUT

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.

```
void *getFreq(void *args)
{
        long freq = *((long*)args);

        while(1){

                pthread_mutex_lock(&lock);
                long new_freq = get_freq();

                if ( new_freq != freq )
                {
                        freq = new_freq;


                }
                pthread_mutex_unlock(&lock);


                }

}
```

3.3 Thread LOGGING

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".

```
void *save_time(void *args)
{
        tp.tv_sec = 0;
        tp.tv_nsec = 0;
        while(1){

                if(tp.tv_sec != T.tv_sec || tp.tv_nsec != T.tv_nsec)
```

```
                {
                FILE *file;
                file = fopen("time_and_interval.txt","a+");
                //file1 = fopen("100.txt","a+");
                long sub_sec;
                long sub_nsec;
                if(T.tv_nsec < tp.tv_nsec){
                        sub_nsec = 1000000000 + T.tv_nsec - tp.tv_nsec;
                        sub_sec = T.tv_sec - tp.tv_sec - 1;

                }
                else
                {
                        sub_nsec =  T.tv_nsec - tp.tv_nsec;
                        sub_sec = T.tv_sec - tp.tv_sec;
                }

                fprintf(file,"\n%ld.%09ld\n%ld.%09ld",T.tv_sec,T.tv_nsec,sub_sec,sub_nsec);
                //fprintf(file1,"\n%ld.%09ld",sub_sec,sub_nsec);
                tp.tv_sec = T.tv_sec;
                tp.tv_nsec = T.tv_nsec;
                fclose(file);
                //fclose(file1);

                }

        }

}
```

Hàm main

```
int main(int argc, char const *argv[])
{
        freq = get_freq();
        pthread_t SAMPLE;
        pthread_t INPUT;
        pthread_t LOGGING;
        int a1, a2, a3;
        T.tv_sec = 0;
        T.tv_nsec = 0;
        pthread_mutex_init(&lock, NULL);
        a1 = pthread_create(&INPUT,NULL,getFreq,&freq);
        a2 = pthread_create(&SAMPLE, NULL,getTime,&freq);
        a3 = pthread_create(&LOGGING,NULL,save_time,&T);
        pthread_join(INPUT,NULL);
        pthread_join(SAMPLE,NULL);
        pthread_join(LOGGING,NULL);
        pthread_mutex_destroy(&lock);
        return 0;
}
```

3.4 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.

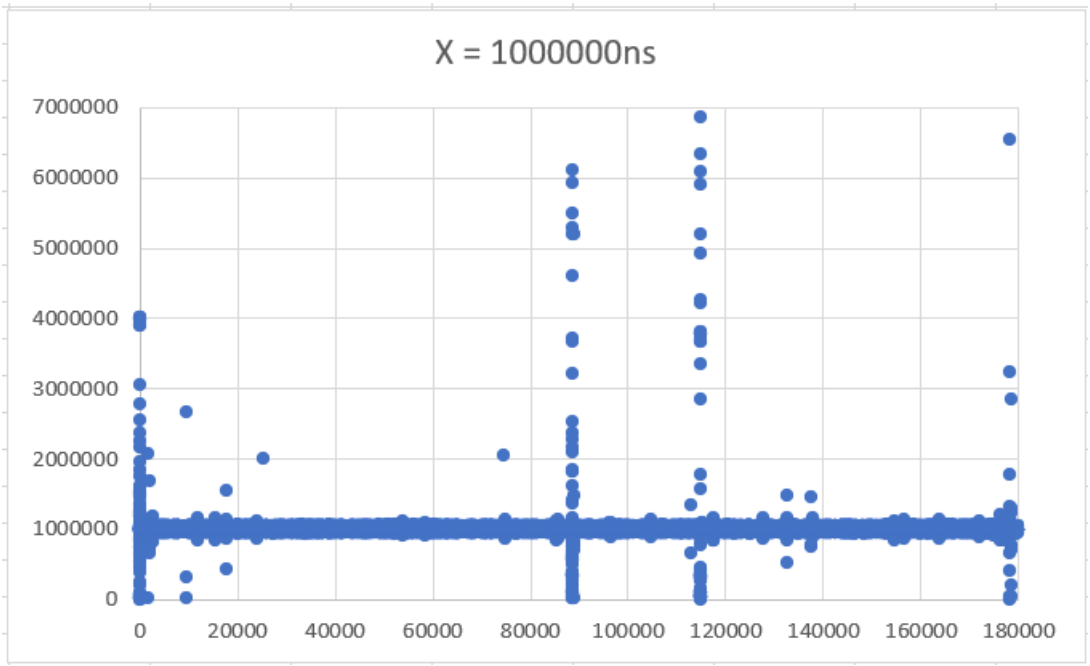
```
#!/bin/bash

gcc -pthread -o bai1 bai1.c
echo "start"
echo "1000000">freq.txt
timeout 60s ./bai1
echo "100000">freq.txt
timeout 60s ./bai1
echo "10000">freq.txt
timeout 60s ./bai1
echo "1000">freq.txt
timeout 60s ./bai1
echo "100">freq.txt
timeout 60s ./bai1
echo "end"
```

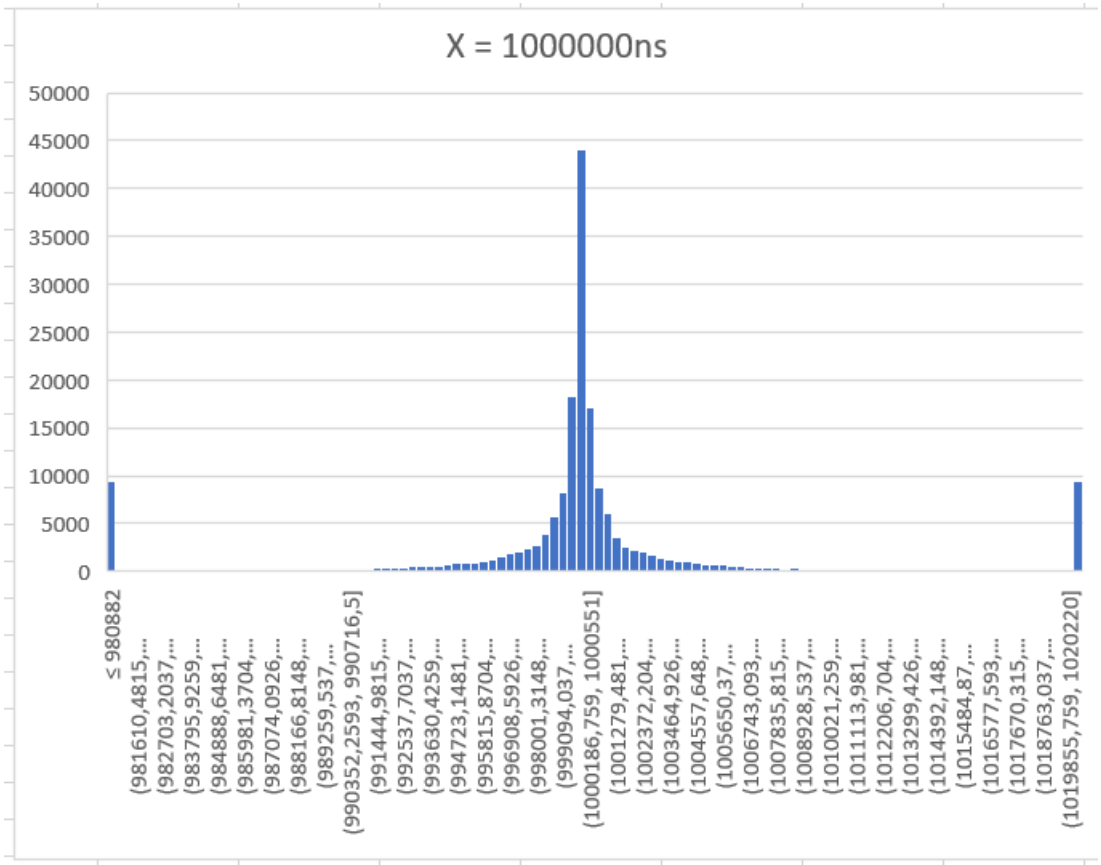
3.5 Chạy shell script + chương trình C trong vòng 5 phút, sau đó dừng chương trình C.

4. Kết quả và phân tích kết quả

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.

4.1 Với chu kì X = 1000000ns



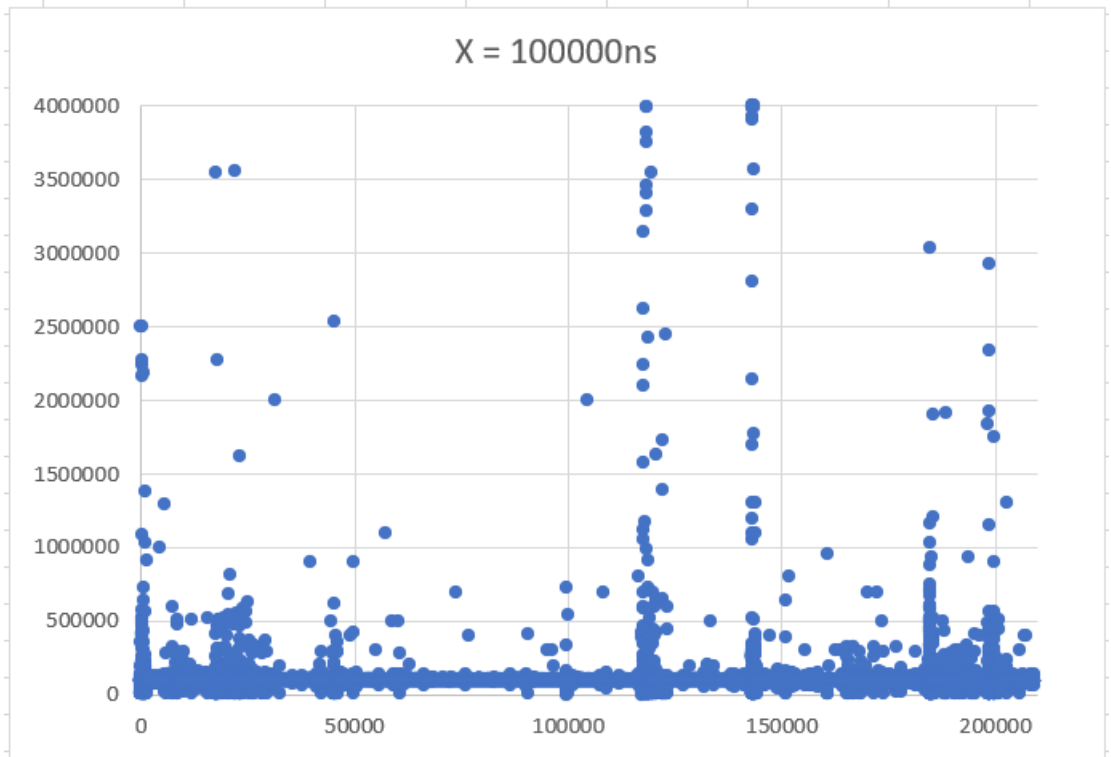


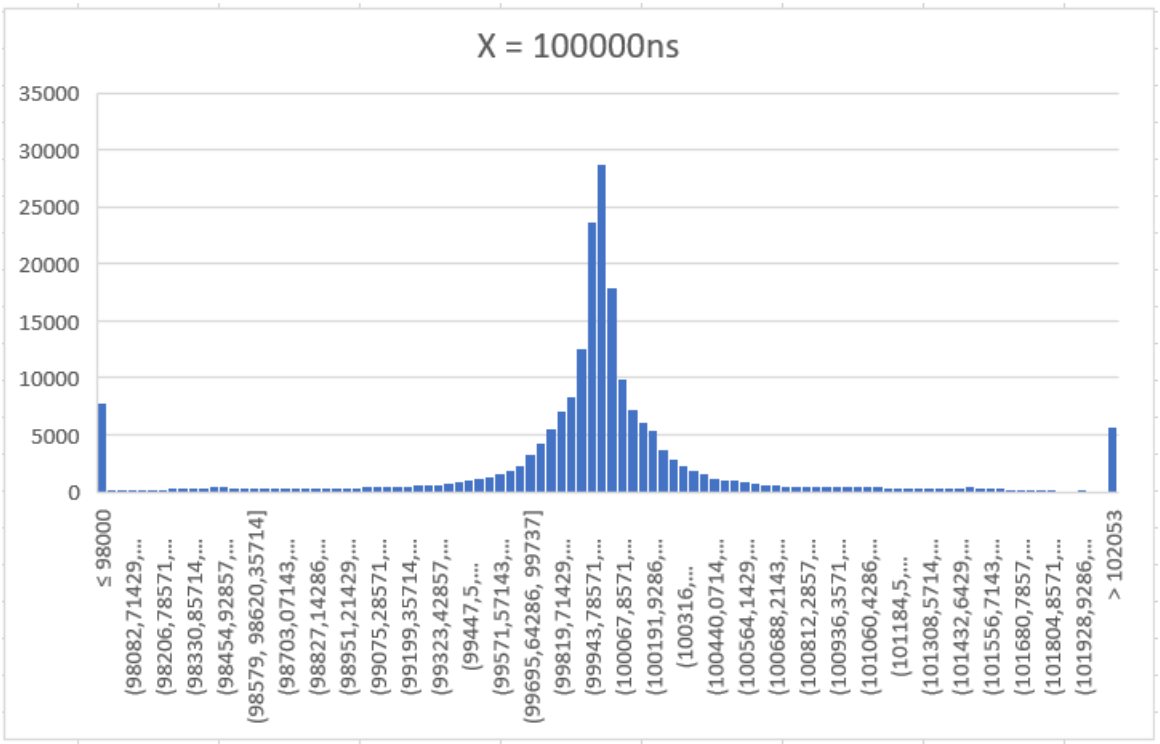
Trong 180000 mẫu lấy được thì có khoảng 44000 mẫu nằm trong khoảng (999094, 1000186) ns chiếm tỉ lệ mẫu cao nhất.

Khoảng 9000 mẫu <= 980882ns và khoảng 9000 mẫu >=1019855

Các mẫu chủ yếu nằm lân cận 1000000.

4.2 Với chu kì X = 100000ns



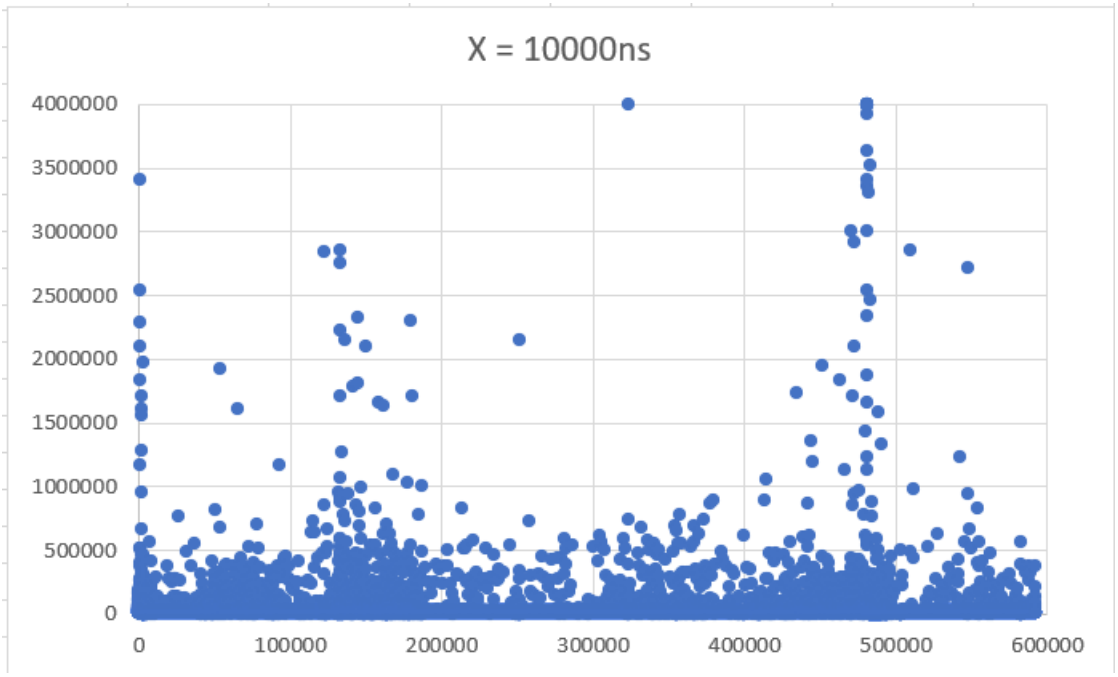


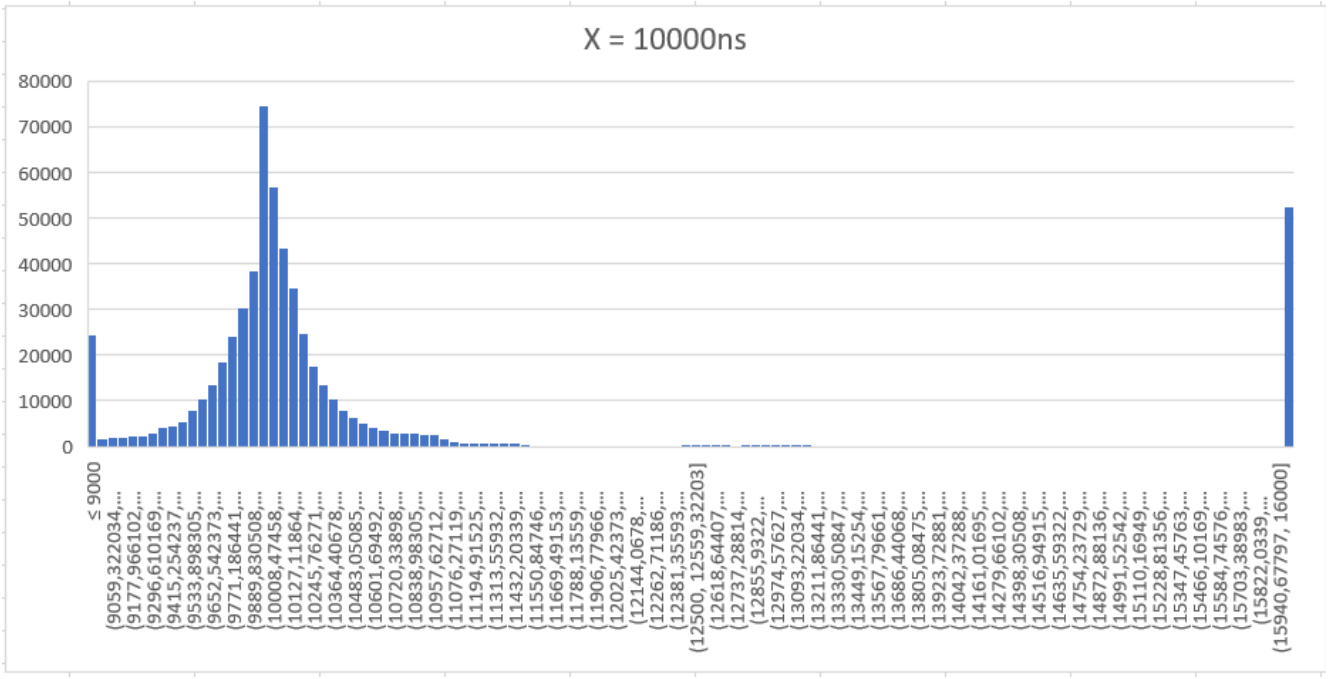
Trong 210000 mẫu, có khoảng 28000 mẫu trong khoảng (99943, 100067) chiếm tỉ lệ mẫu cao nhất.

Có khoảng 7000 mẫu <=98000 và hơn 5000 mẫu >102053

Các mẫu chủ yếu chạy lân cận quanh 100000.

4.3 Với chu kì X = 10000ns

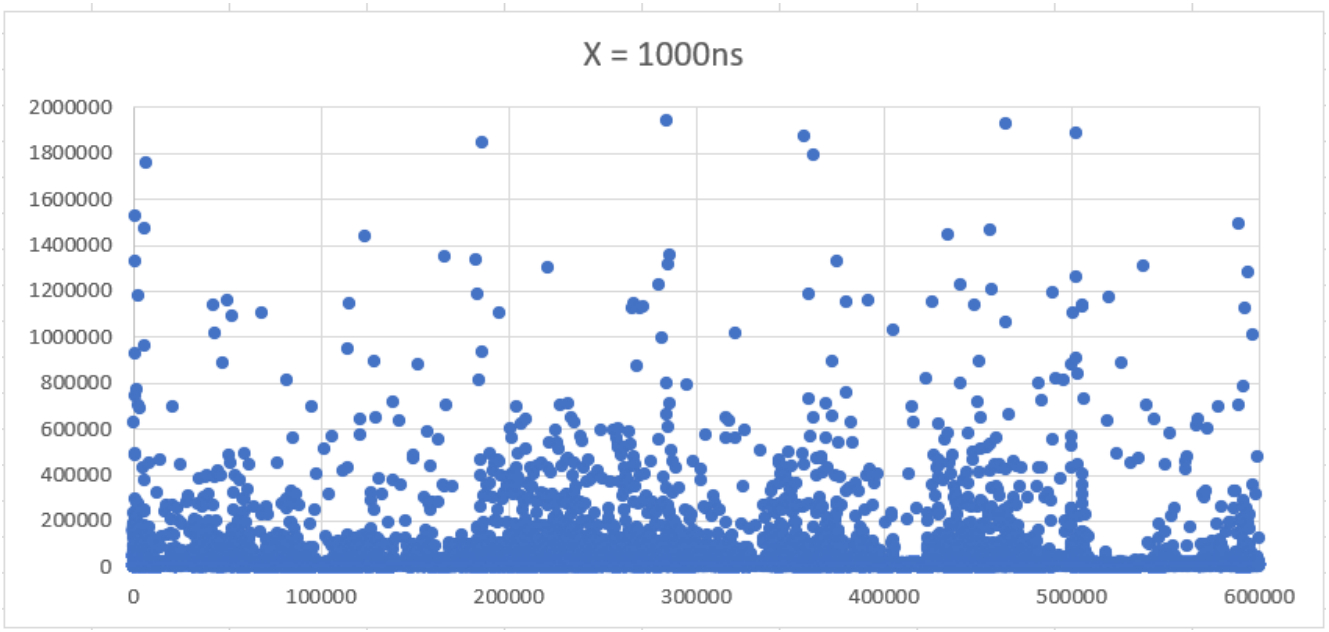


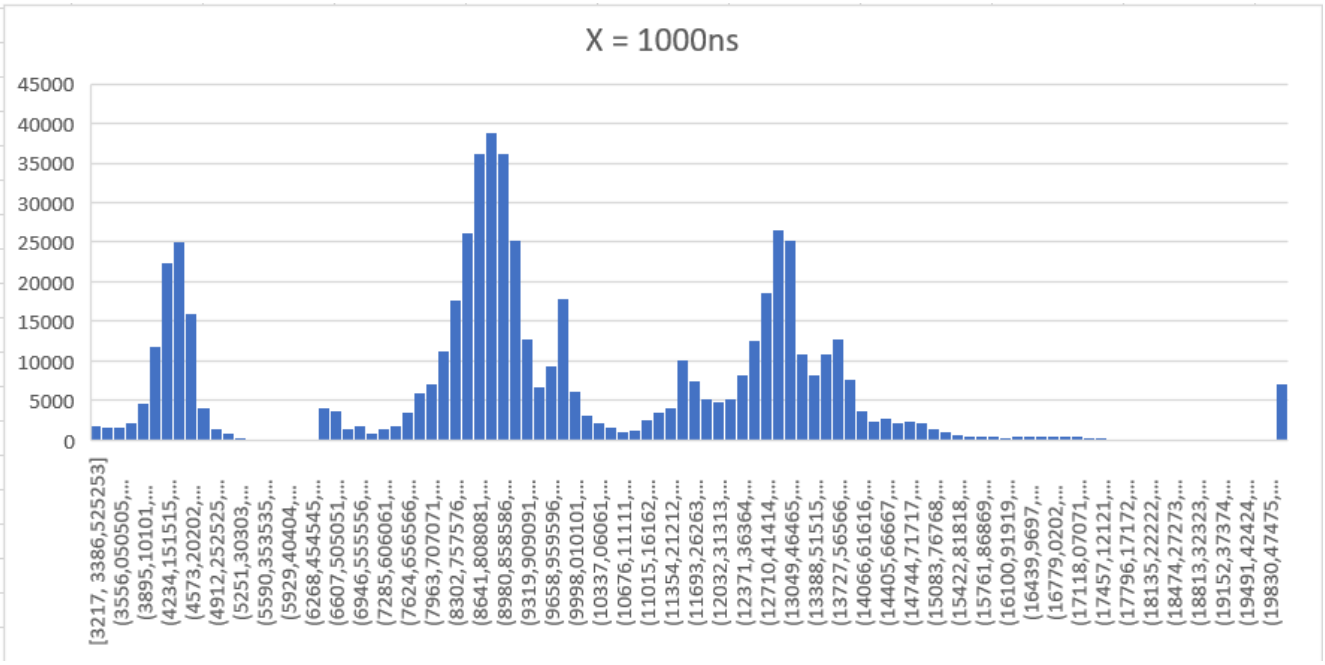


Trong khoảng 600000 mẫu, có 74455 nằm trong khoảng (9889,10008) chiếm tỉ lệ mẫu cao nhất.

Có 24423 mẫu <=9000 và có 52374 mẫu >15940(chiếm tỉ lệ số mẫu cao thứ 2).

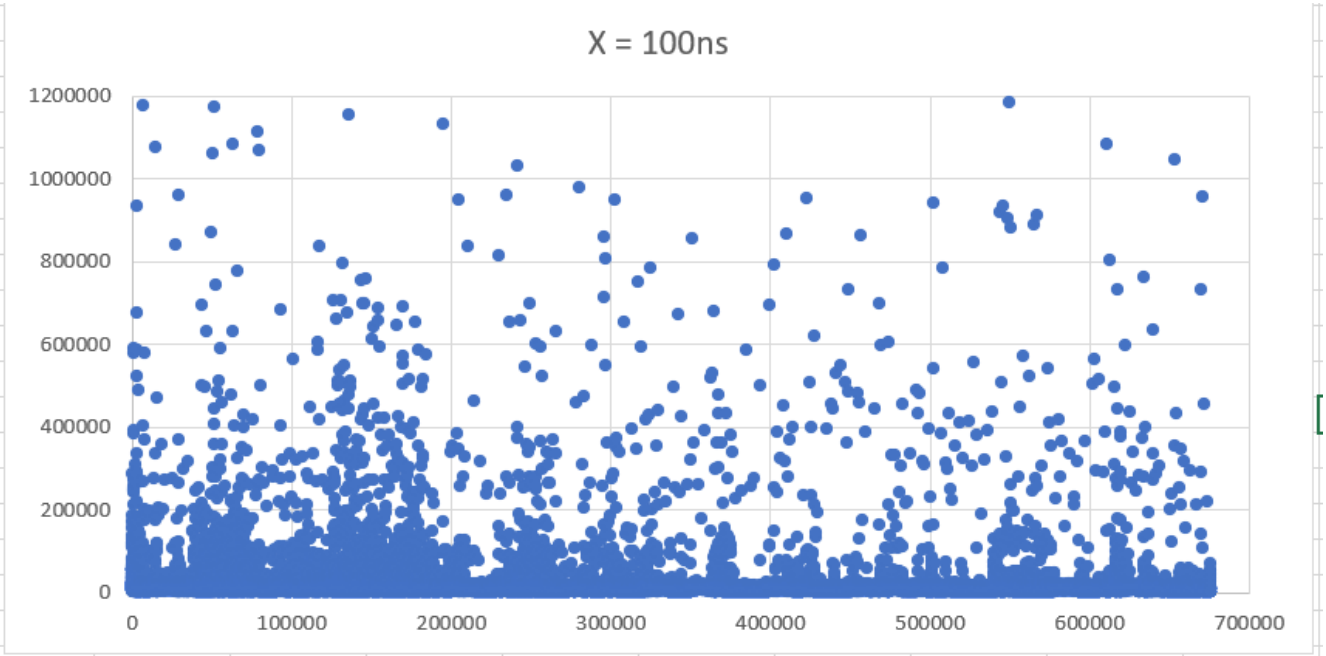
4.4 Với chu kì X = 1000ns

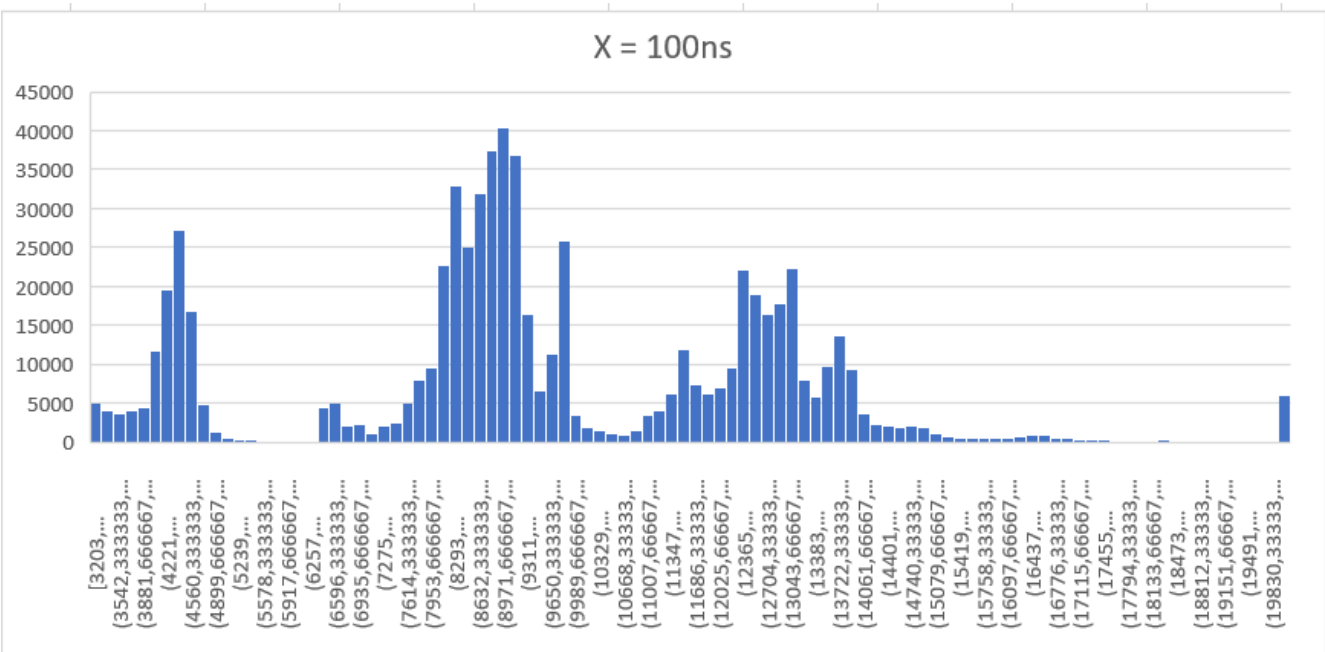




Với X = 1000ns, ta lấy khoảng 600000 mẫu. Đồ thị histogram này đã khác với 3 đồ thị ở trên nó. 3 đồ thị ứng với X = {1000000, 100000, 10000} đều là dạng 1 đỉnh rồi thoải ra 2 bên. Với X = 1000 ta thấy có 3 đỉnh, các cột khác đều thoải đều sang 2 bên ứng với mỗi đỉnh. 3 đỉnh cực trị lần lượt là 25036 mẫu nằm trong khoảng (4234, 4573), 38853 mẫu nằm trong khoảng (8641,8980), 26548 mẫu nằm trong khoảng (12710, 13049).

4.5 Với chu kì X = 100ns





Với X = 100ns, ta lấy khoảng 680000 mẫu. Đồ thị histogram cũng có 3 cực trị. Các mẫu lấy được lệnh quá nhiều so với 100.

5. Kết luận

Với X = {1000000, 100000} thì giá trị interval lệch khá nhỏ so với X, đặc biệt là X = 1000000ns

Với X = 10000 thì interval lệch so với X đã nhiều hơn nhưng vẫn chấp nhận được

Với X = 1000 thì interval lệch rất nhiều so với X

Với X = 100 thì thất bại, hầu như mọi interval đều lớn hơn 3000

Khó khăn gặp phải:

• Chưa quen với Linux Programming Assignment

• Kết quả thu được chưa được như kì vọng, giá trị interval còn lệnh nhiều so với X

• Chưa tối ưu được thuật toán