BÁO CÁO
LINUX PROGRAMMING ASSIGMENT

Nhóm thực tập nhúng - dự án gNodeB 5G VHT

Sinh viên: Cấn Quang Thịnh - MSSV:19020634

MỤC LỤC

I. TRIỂN KHAI VÀ PHÂN TÍCH KẾT QUẢ

1. Viết chương trình C trên Linux chạy 3 thread SAMPLE, LOGGING, INPUT.

1.1 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.

```
void * sam_func(void *arg) {
        //chờ

       clock_gettime(CLOCK_MONOTONIC, &request);

        while(1) {

      //  printf("sam count %d\n",count_sam);

        pthread_mutex_lock(&mtx);

        while(input_flag == 0) {
              pthread_cond_wait(&condition_input,&mtx);
        }

        pthread_mutex_unlock(&mtx);

        request.tv_nsec += T;

        if(request.tv_nsec > 1000*1000*1000) {
              request.tv_nsec -= 1000*1000*1000;
              request.tv_sec++;
        }
        clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &request, NULL);
        clock_gettime(CLOCK_MONOTONIC, &rtc);
        pthread_mutex_lock(&mtx);

        input_flag = 0;
        sample_flag = 1;
       // count_sam ++;

        pthread_cond_signal(&condition_sample);

        pthread_mutex_unlock(&mtx);
        }

    return NULL;

}
```

1.2 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 * in_func(void *arg) {



    while(1) {

    // printf("in count ->%d\n",count_in);
     fp = fopen("freq.txt","r");
     char buff[100];
     fgets(buff,sizeof(buff),fp);
     char *eptr;
     T = strtol(buff,&eptr,10);

     fclose(fp);


     pthread_mutex_lock(&mtx);
     input_flag = 1;
     sample_flag = 0;
    // count_in ++;
     pthread_cond_signal(&condition_input);
     pthread_mutex_unlock(&mtx);

    }
     return NULL;

}
```

1.3 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    *log_func (void* arg) {
        while(1) {
        pthread_mutex_lock(&mtx);
        while(sample_flag == 0) {
                pthread_cond_wait(&condition_sample,&mtx);
        }
        input_flag = 0;
        sample_flag = 0;
        pthread_mutex_unlock(&mtx);
        if(rtc.tv_nsec != ot.tv_nsec ||rtc.tv_sec != ot.tv_sec ) {
        file = fopen("time_and_interval.txt","a+");
        if(file == NULL) {
                printf("ko mo dc file\n");
                return NULL;
        }
        long diff_sec = (long)rtc.tv_sec - (long)ot.tv_sec ;
        long diff_nsec;
        if(rtc.tv_nsec > ot.tv_nsec) {
                diff_nsec = rtc.tv_nsec - ot.tv_nsec;
        }
        else {
                diff_nsec = 1000000000 - ot.tv_nsec + rtc.tv_nsec;
                diff_sec = diff_sec - 1;
        }
        if(ot.tv_nsec != 0) {
                //fprintf(file, "%ld.%09ld %ld.%09ld\n",rtc.tv_sec,rtc.tv_nsec,diff_sec,diff_nsec);
                //fprintf(file, "%ld,%09ld\n",diff_sec,diff_nsec);
                fprintf(file, "%ld\n",diff_nsec);
        }
        fclose(file);
        ot.tv_nsec = rtc.tv_nsec;
        ot.tv_sec = rtc.tv_sec;
        }
        pthread_mutex_lock(&mtx);
        input_flag = 1;
        sample_flag = 0;
        pthread_cond_signal(&condition_input);
        pthread_mutex_unlock(&mtx);
        }
        return NULL;
}
```

2. 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.

```
#! /usr/bin/bash

i=0
val=1000000
# while [ $i -lt 5 ]
# do
    > time_and_interval.txt
    echo  "$val" > "freq.txt"
    timeout 60s ./last_code
#     ((val/=10))
#     ((i++))

# done
```

3. Thực hiện chạy shell script + chương trình C

Trong phần này, ta sẽ cho chạy chương trình C và shell script trong vòng 5 phút, và sau đó dừng chương trình C.

4. Khảo sát giá trị interval

Trong phần cuối này, ta thực hiện khảo sát file “time_and_interval.txt”.Bằng cách vẽ đồ thị giá trị và biểu đồ histogram của interval đối với mỗi giá trị chu kỳ X đồng thời đánh giá và đưa ra kết luận. Trong phần này sẽ sử dụng gnuplot để mình họa các kết quả thu được.

II. LÝ THUẾT SỬ DỤNG

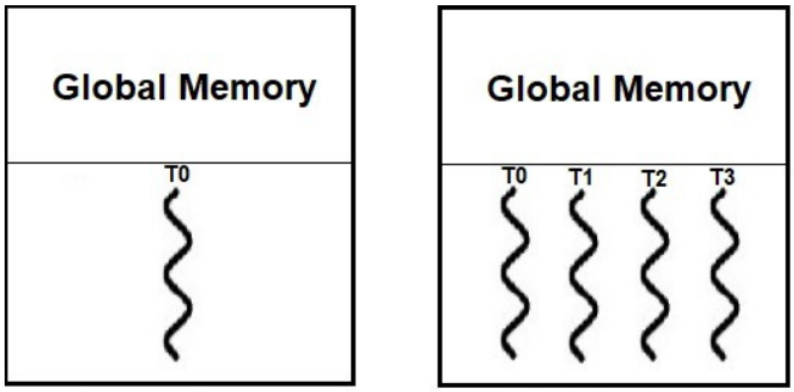
1. Multithread

Thread là một cơ chế cho phép một ứng dụng thực thi đồng thời nhiều công việc (multi-task).

Thread là một thành phần của tiến trình, một tiến trình có thể chứa một hoặc nhiều thread. Hệ điều hành Unix quan niệm rằng mỗi tiến trình khi bắt đầu chạy luôn có một thread chính (main thread);

Nếu không có thread nào được tạo thêm thì tiến trình đó được gọi là đơn luồng (single-thread), ngược lại nếu có thêm thread thì được gọi là đa luồng (multi-thread).

Các thread trong tiến trình chia sẻ các vùng nhớ toàn cục (global memory) của tiến trình bao gồm initialized data, uninitialized data và vùng nhớ heap.



Tiến trình single-thread Tiến trình multi-thread

2.Sleep

Hàm nanosleep() thực hiện nhiệm vụ tương tự như sleep() , nhưng có thể đạt được độ chính xác nanosecond

```
int nanosleep(const struct timespec *request, struct timespec *remain);
```

Giống như nanosleep() thì clock_nanosleep() sẽ tạm dừng quá trình gọi cho đến khi một khoảng thời gian cụ thể trôi qua hoặc một tín hiệu đến.

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

3. Thread synchronization

Ta sử dụng khóa mutex và biến cond var để đồng bộ hóa tiến trình:

- Mutex cung cấp cơ chế khóa (lock) một đối tượng để đảm bảo tại một thời điểm chỉ có một thread truy cập vào các biến dùng chung

Khởi tạo mutex:

```
Pthread_mutex_t mutex =  PTHREAD_MUTEX_INITIALIZER;
```

Thực hiện việc khóa : `pthread_mutex_lock(pthread_mutex_t *mutex);`

Thực hiện mở khóa : `pthread_mutex_unlock(pthread_mutex_t *mutex);`

- Biến điều kiện cho phép một thread đăng ký đợi (đi vào trạng thái ngủ) cho đến khi một thread khác gửi một signal đánh thức nó dậy để chạy tiếp

Khởi tạo biến điều kiện : pthread_cond_t cond_var=PTHREAD_COND_INTIALIZER

Hàm thực hiện chờ một thread : pthread_cond_wait(<condition variable>,<mutex>);

Hàm đánh thức thread : pthread_cond_signal(<condition variable>);

4. Shell script

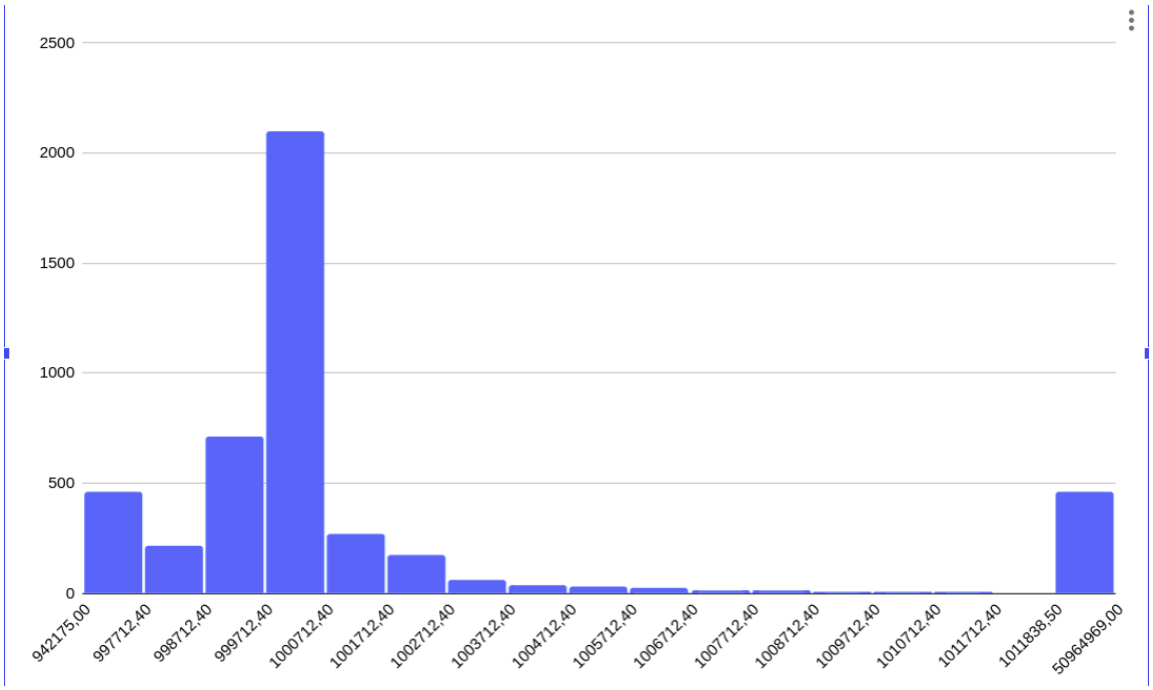
Shell là một chương trình thông dịch lệnh của một hệ điều hành, cung cấp khả năng tương tác với hệ điều hành bằng cách gõ từng lệnh ở chế độ dòng lệnh, đồng thời trả lại kết quả thực hiện lệnh lại cho người sử dụng.

Shell script cung cấp tập hợp các lệnh đặc biệt mà từ đó có thể tạo nên chương trình. Shell được bắt đầu khi người dùng đăng nhập hoặc khởi động terminal.

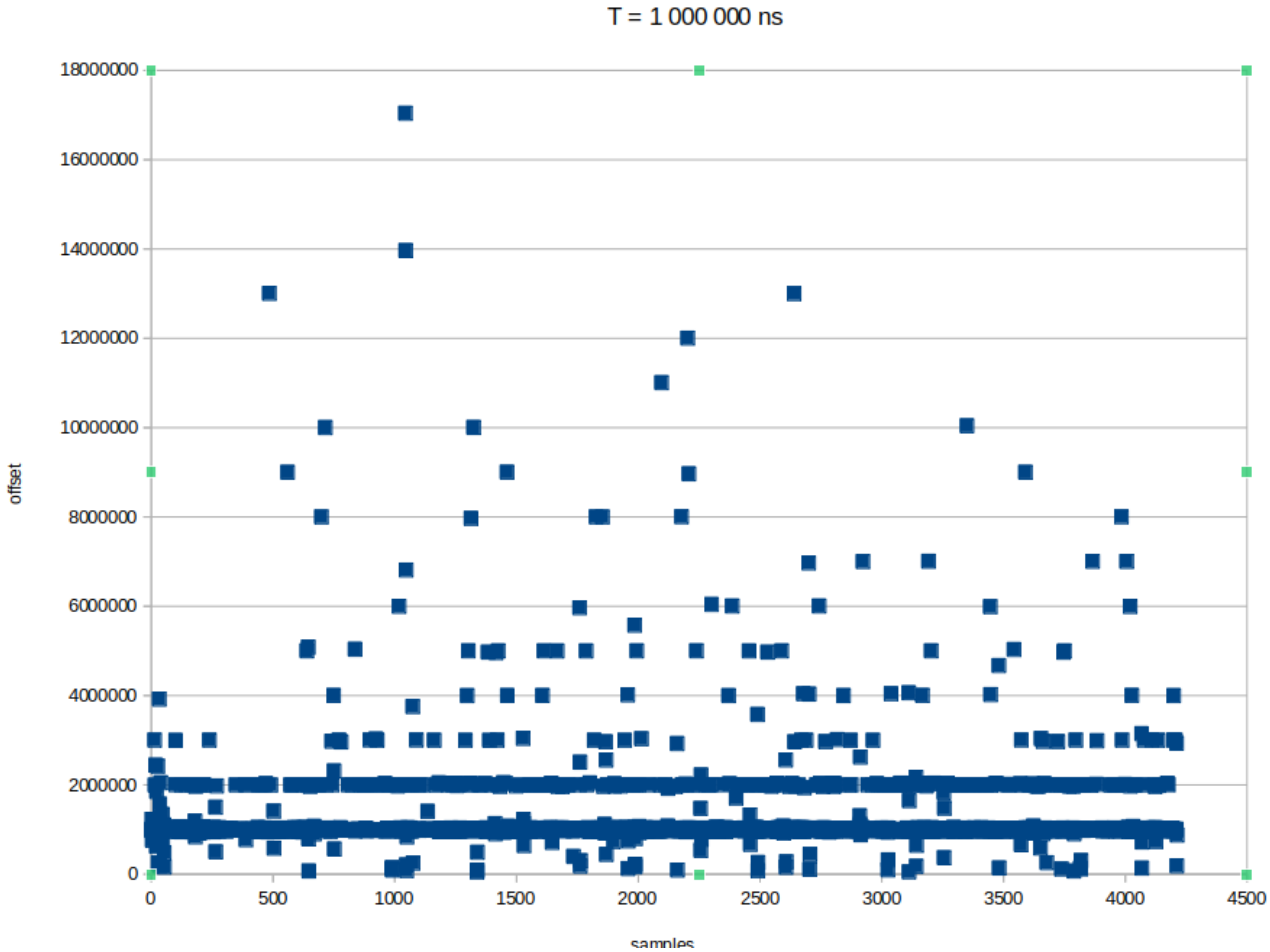
III. Kết Quả

1. Khi T = 1000 000 ns.

1.1 Biểu đồ histogram

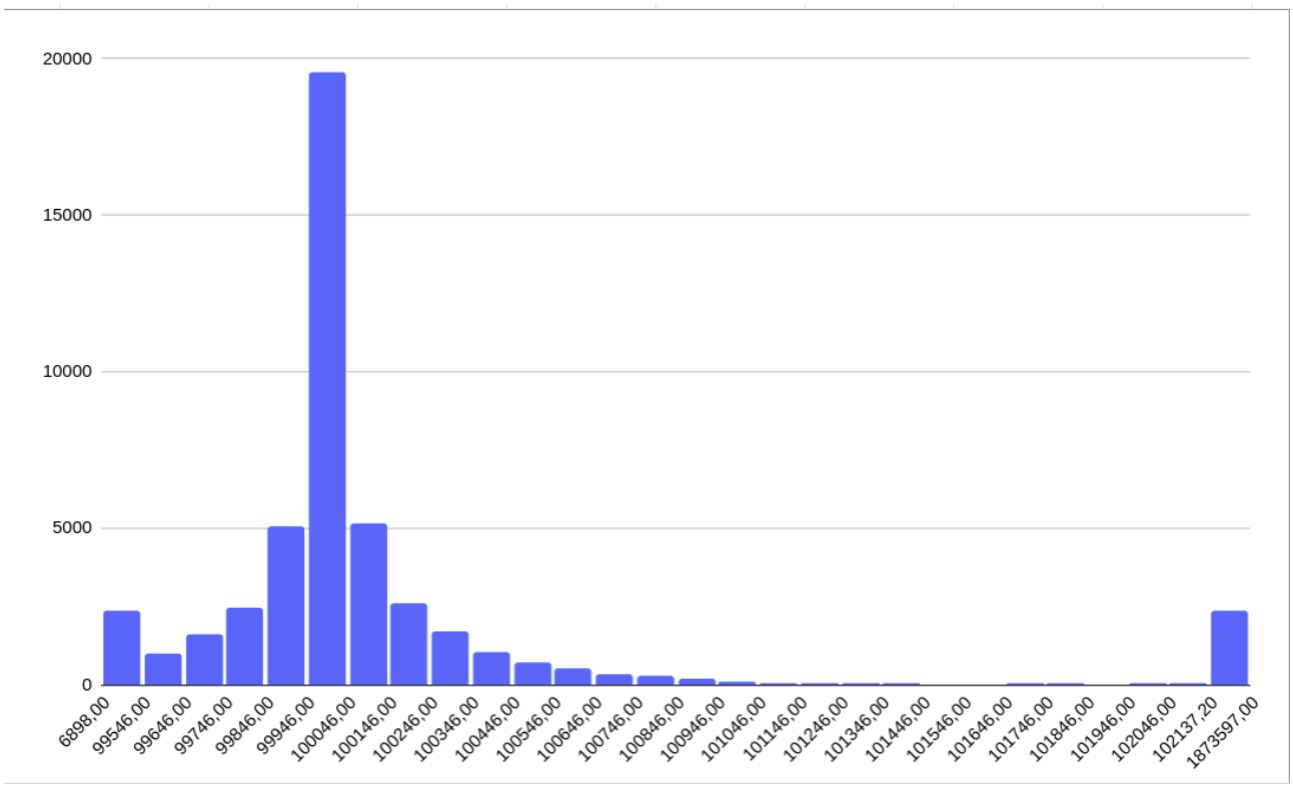


1.2 Biểu đồ value

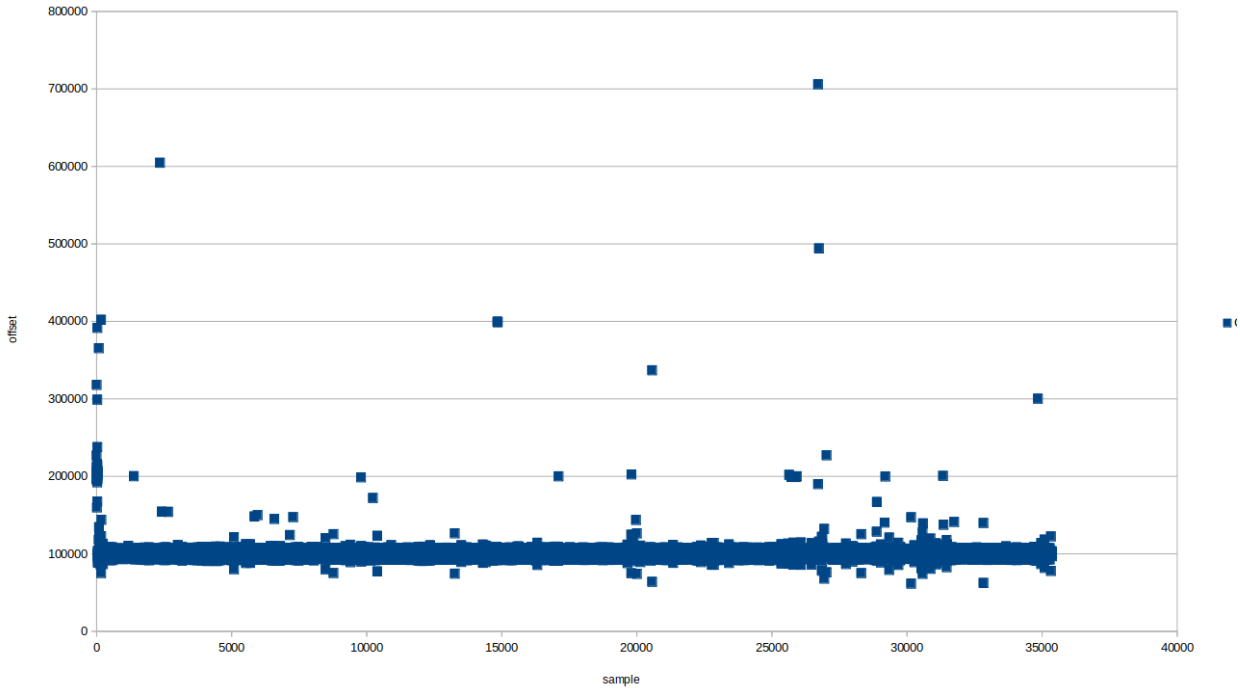


2. Khi T = 100 000 ns.

2.1 Biểu đồ histogram

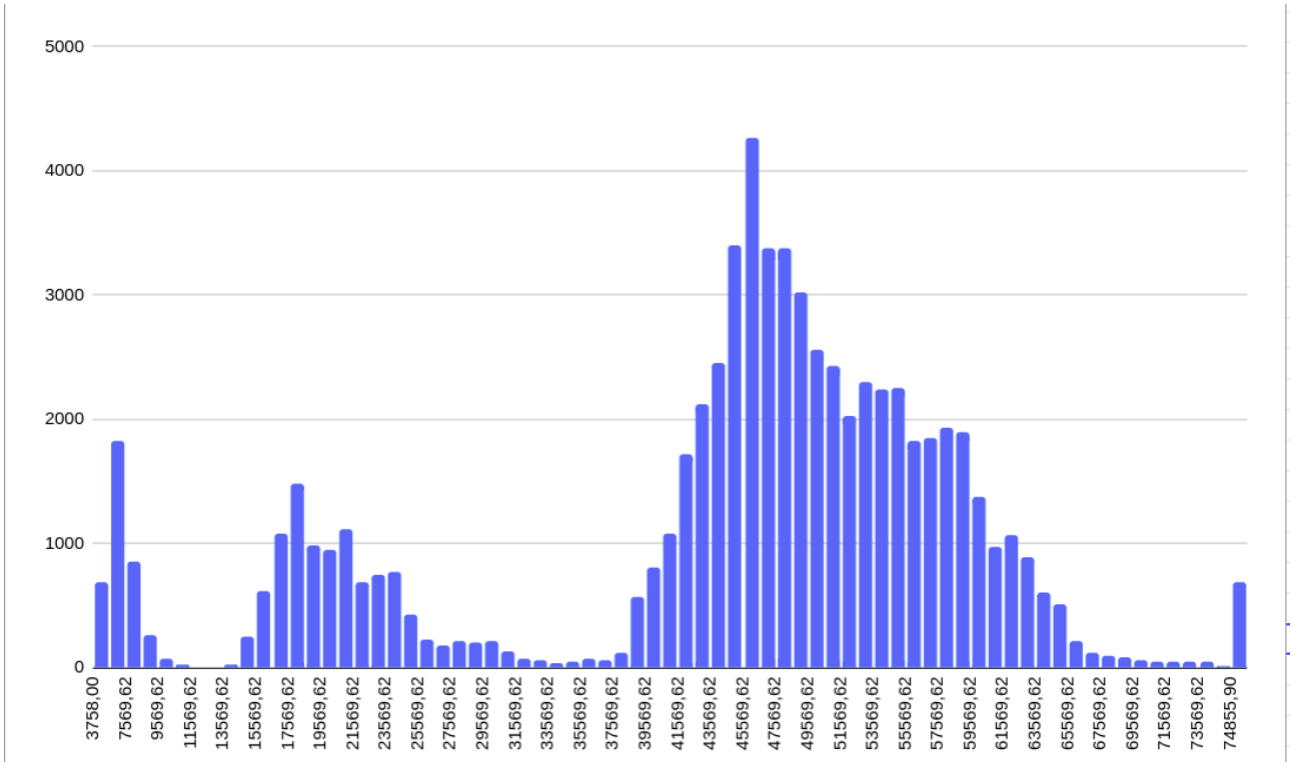


2.2 Biểu đồ value

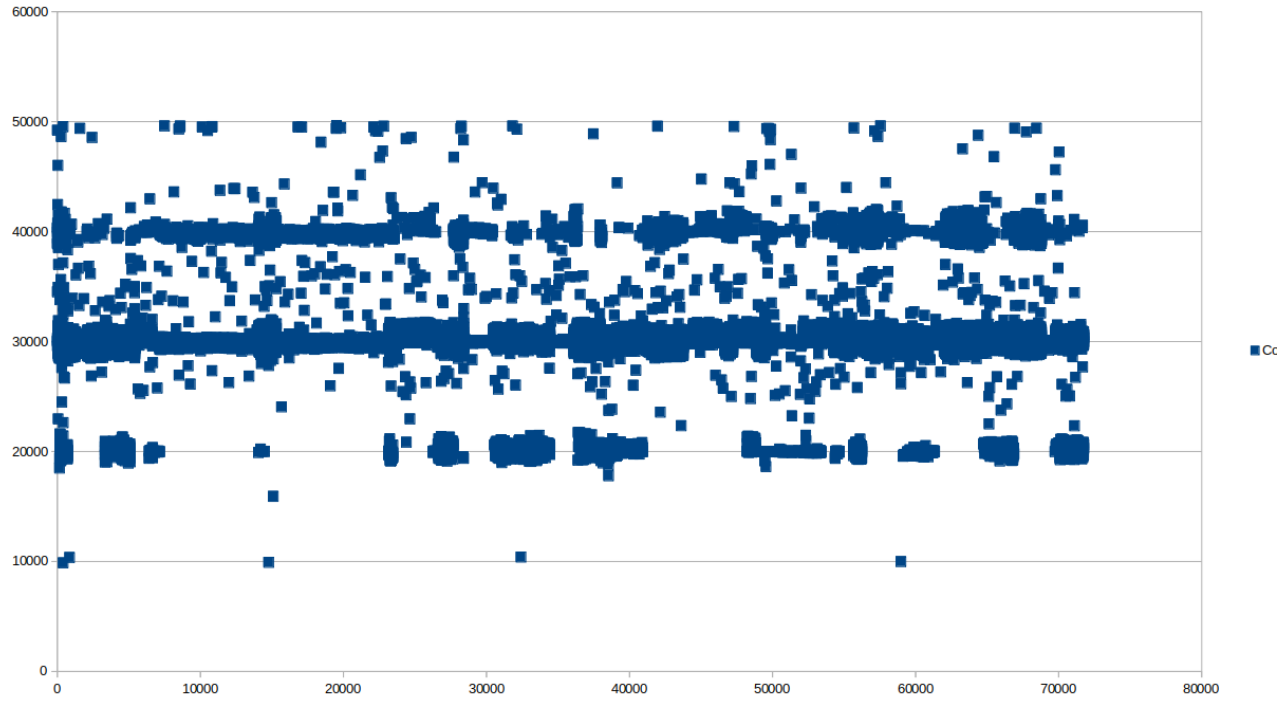


3. Khi T = 10 000 ns.

3.1 Biểu đồ histogram

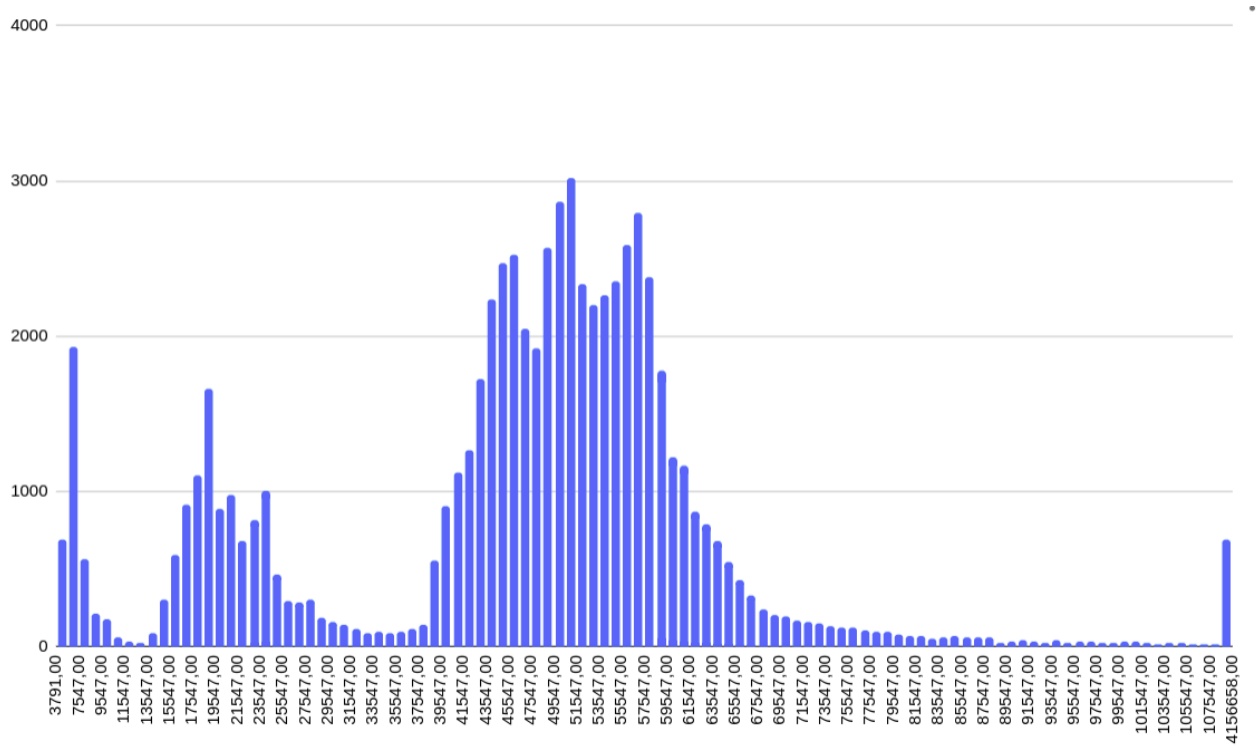


3.2 Biểu đồ value

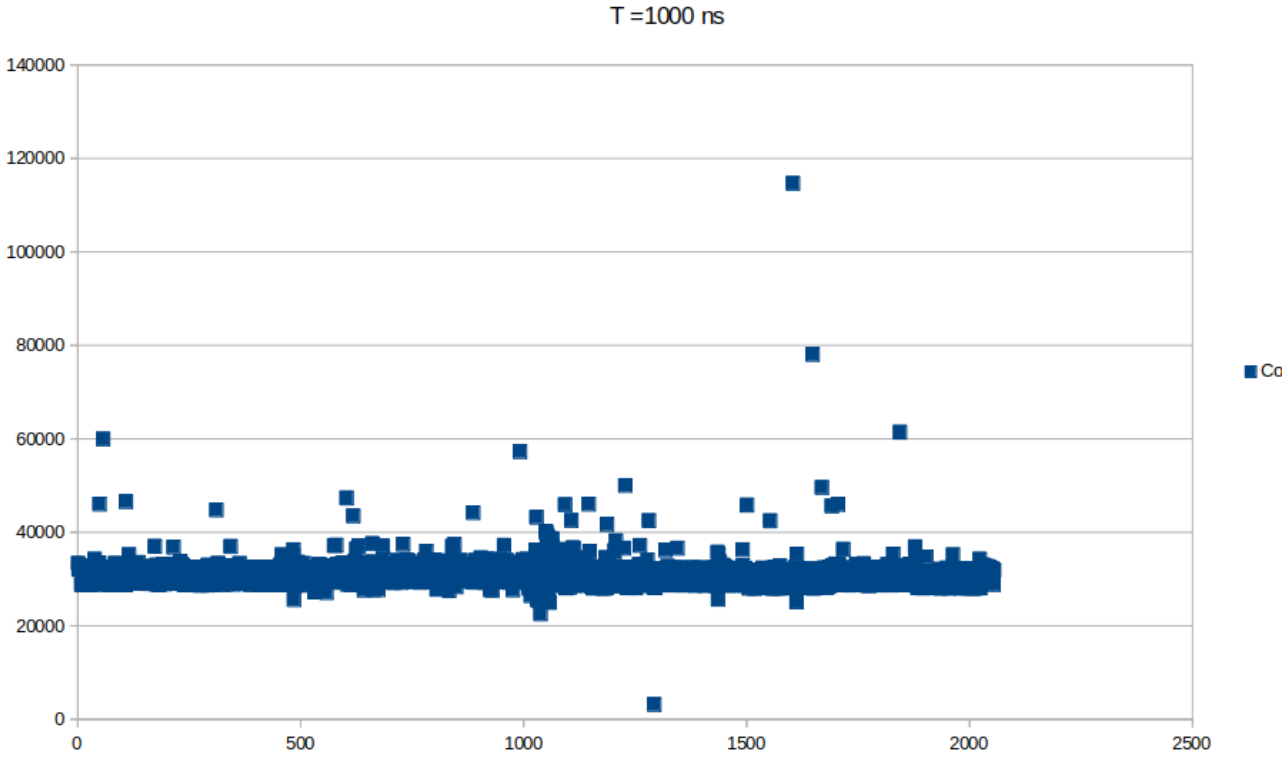


4. Khi T = 1000 ns.

4.1 Biểu đồ histogram

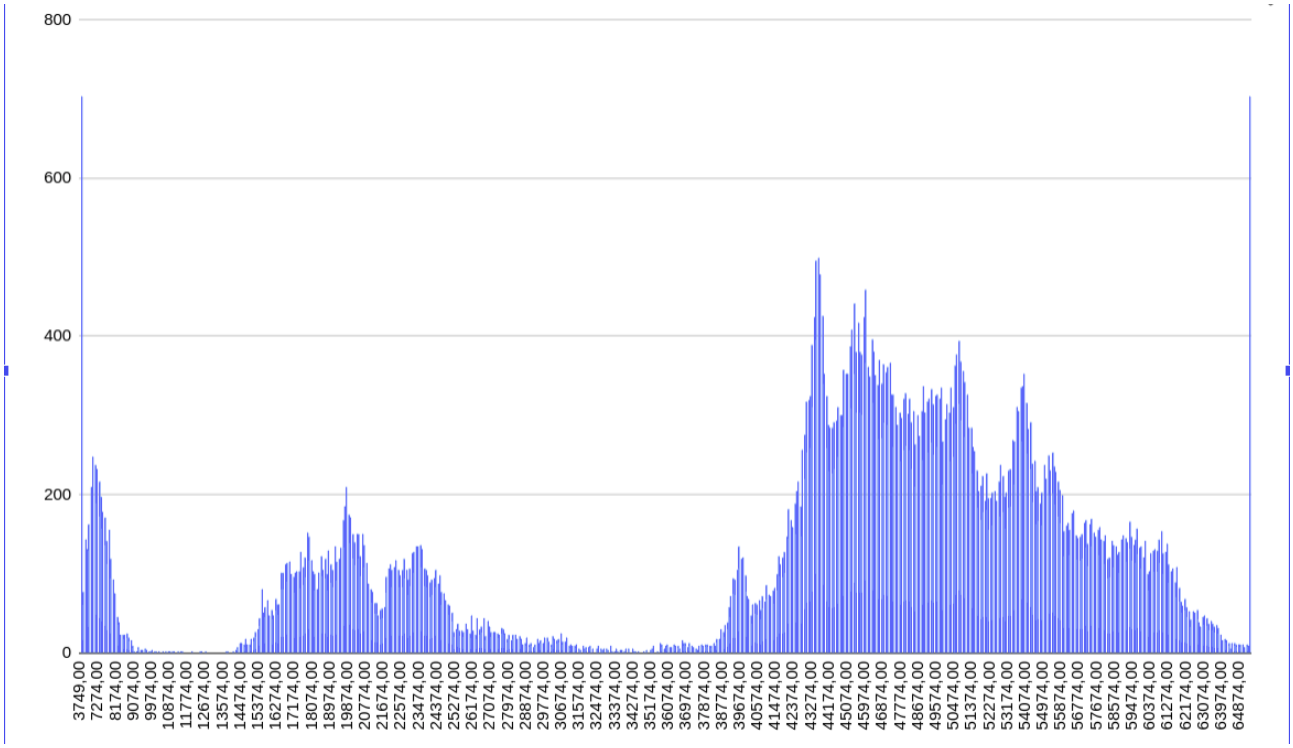


4.2 Biểu đồ value

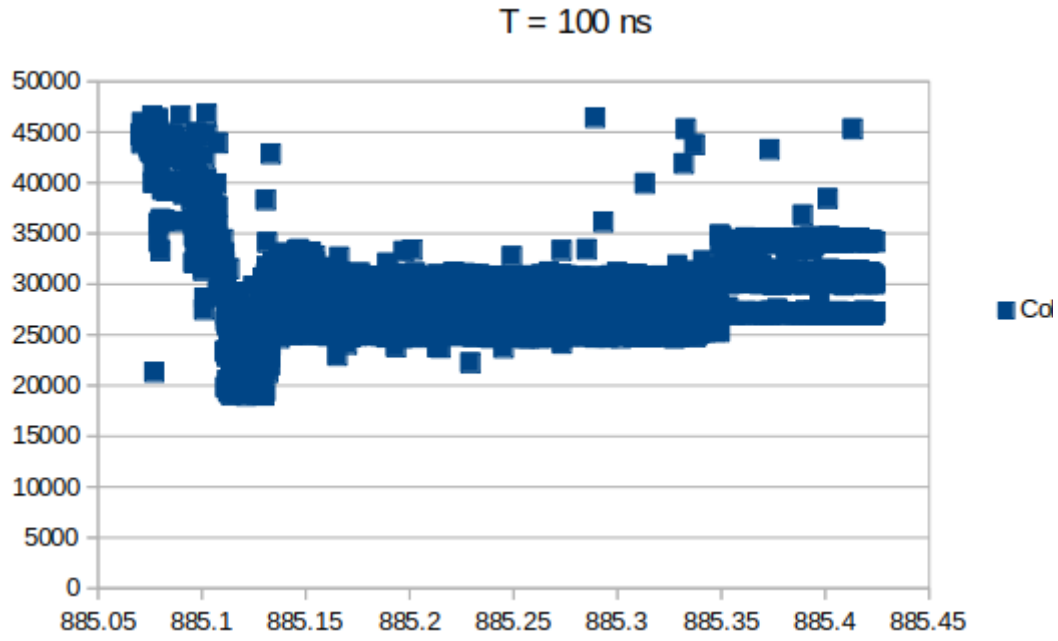


5. Khi T = 100 ns.

5.1 Biểu đồ histogram



5.2 Biểu đồ value



IV. Kết Luận

+ T = 1 000 000 ns thì giá trị offet giao động chủ yếu tập trung ở vùng 1 000 000 ns
+ T = 100 000 ns thì giá trị offset giao động chủ yếu tập trung ở vùng 100 000 ns
+ T = 10 000 ns thì giá trị offset giao động chủ yếu ở vùng 45 000 ns
+ T = 1000 ns thì giá trị offset giao động chủ yếu ở vùng 49 000 ns
+ T = 100 ns thì giá trị offet giao động chủ yếu ở vùng 44 000 ns

=> khi T < 100 000 ns thì giá trị offset nhận được bị sai lệch rất nhiều so với T mong đợi
đặc biệt T =100 ns thì offset còn không đạt được ngưỡng trăm ns

=> Code chưa tối ưu ,dù đã xử lý tràn trong việc dùng cock_nanosleep,sử dụng khóa mutex, và cond var để tạo thứ tự chạy cho các luồng nhưng kết quả vẫn không như mong đợi

+ Em nghĩ do mình dùng các khóa mutex,và cond var để đồng bộ 3 luồng đã tạo thêm thời gian chờ để thực hiện các luồng