BÁO CÁO

LINUX PROGRAMMING ASSIGMENT

Nhóm thực tập Nhúng – dự án gNodeB 5G VHT

Sinh viên: Vũ Thị Hồng Nhung 19021498

Hà Nội, 15/7/2022

MỤC LỤC

MỤC LỤC HÌNH ẢNH

I. PHẦN GIỚI THIỆU

1. Đặt vấn đề

Linux là một hệ điều hành máy tính mã nguồn mở và tự do dạng Unix-like được xây dựng trên nền của nhân Linux (hay còn gọi là Linux kernel).

2. Các lý thuyết liên quan

Trong phần này, ta sẽ nhắc đến các lý thuyết về lập trình multi thread, thread synchronization, sleep, shell script ...

Nếu trong một tiến trình có thể có nhiều nhanh thực hiện, các nhánh này có thể thực hiện độc lập và song song với nhau, khi đó ta gọi mỗi nhánh thực hiện trên là một luồng (thread). Thread là một thành phần của tiến trình, một tiến trình có thể chứa một hoặc nhiều thread và thread là một chuỗi điều khiển trong một tiến trình. Vậy nên nếu không có thread nào được tạo thêm thì tiến trình đó được gọi là **single-thread**, ngược lại nếu có thêm thread thì được gọi là **multi-thread**.

Ta thấy thao tác tạo ra thread nhanh hơn so với tạo triến trình. Hơn thế nữa, thời gian hệ thống chuyển từ thread này sang thread khác nhanh hơn từ tiến trình này sang tiến trình khác. Và giao tiếp (đồng bộ) giữa các thread trong cùng một tiến trình có thể thực hiện dễ dàng hơn so với giữa các tiến trình do các thread cùng nhau chia sẻ tài nguyên của tiến trình đó. Bên cạnh đó, việc thiết kế, xây dựng multi-thread cần tỉ mỉ cẩn thận bởi các thread có khả năng sử dụng tài nguyên chung của tiến trình và gỡ rối một tiến trình đa luồng phức tạp hơn so với tiến trình đơn luồng.

Để xây dựng tiến trình đa luồng trên Linux ta cần sử dụng thư viện pthread, tức là thêm `#include<pthread>` và khi biên dịch cần thêm tham số `-lpthread`.

– Tạo thread:

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

– Kết thúc thread:

```
void pthread_exit(void *retval);
```

– Chờ một thread kết thúc

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

– Đồng bộ thread bằng mutex

Mutex cung cấp cơ chế khóa (lock) một đối tượng để đảm bảo tại một thời điểm chỉ có một thread thực hiện đoạn mã của mình. Có hai cách khởi tạo một mutex:

- Khởi tạo tĩnh:

```
pthread_mutex_t mutex =  PTHREAD_MUTEX_INITIALIZER;
```

- Khởi tạo động:

```
int pthread_mutex_init(pthread_mutex_t *mutex, const
pthread_mutexattr_t *attr);
```

Sau khi khởi tạo, mutex được khóa và mở bởi hai hàm dưới đây:

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

Ngoài ra, để giải phóng một biến mutex ta sử dụng hàm:

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

– Thread Synchronization (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.

- Khai báo biến điều kiện: Có hai kiểu là tĩnh và động. Đối cới khai báo tĩnh:

```
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

Còn khai báo động:

```
int pthread_cond_init(pthread_cond_t *cond, const
pthread_condattr_t *attr);
```

Khi không sử dụng ta có thể hủy bằng cách gọi hàm:

```
int pthread_cond_destroy(pthread_cond_t *cond);
```

– Sleep:

Hàm **nanosleep()** thực hiện nhiệm vụ tương tự như **sleep()**, nhưng có một ưu điểm là nó cần chỉ định một khoảng thời gian ngủ (sleep interval):

```
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);
```

– 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.

Vậy thử đặt câu hỏi là tại sao lại cần shell script? Bởi khi có shell script ta có thể tránh các công việc tự động hóa và lặp đi lặp lại, đồng thời giám sát hệ thống và system admins sử dụng shell script để sao lưu thường xuyên.

Shell script có một vàu ưu điểm như sau:

- Viết Shell script sẽ nhanh hơn;
- Lệnh và cú pháp hoàn toàn giống với lệnh được được nhập trực tiếp trong dòng lệnh.

Bên cạnh đó, vẫn còn một vài nhược điểm:

- Tốc độ thực hiện chậm
- Không phù hợp cho các task lớn và phức tạp;
- Dễ xảy ra lỗi tốn kém, một lỗi duy nhất có thể thay đổi lệnh có thể gây hại.

3. Bố cục báo cáo

Bài báo cáo gồm 3 phần chính :

I. Phần giới thiệu

Giới thiệu khái quát về lập trình multithread cùng các kiến thức liên quan như thread synchronization, sleep, shell script sẽ được áp dụng vào bài tập lập trình.

II. Triển khai và phân tích kết quả

Triển khai ý tưởng và cách làm bài tập lập trình C trên Linux. Đồng thời, trong phần này ra sẽ phân tích kết quả và đưa ra kết luận về giá trị interval.

III. Kết luận

Báo cáo tiến độ đồng thời đưa ra những khó khăn gặp phải trong quá trình làm bài.

II. TRIỂN KHAI VÀ PHÂN TÍCH KẾT QUẢ

1. Chương trình C chạy 3 thread SAMPLE, LOGGING, INPUT.

1.1. Thread SAMPLE

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

```
void *sample_func(void *arg)
{
  clock_gettime(CLOCK_REALTIME, &timerqst);

  while(co == 1)
  {
    clock_gettime(CLOCK_REALTIME, &ts);

    long tmp;
    if(timerqst.tv_nsec + frequency > MAX)
    {
```

```
      tmp = timerqst.tv_nsec;
      timerqst.tv_nsec = tmp + frequency - MAX;
      timerqst.tv_sec++;

      if(clock_nanosleep(CLOCK_REALTIME,           TIMER_ABSTIME,
&timerqst, NULL) != 0) co = 0;
      else co = 1;
    }
    else
    {
      timerqst.tv_nsec += frequency;
      if(clock_nanosleep(CLOCK_REALTIME,           TIMER_ABSTIME,
&timerqst, NULL) != 0) co = 0;
      else co = 1;
      }
    }
}
```

Thay vì sử dụng nanosleep(), ta sẽ sử dụng clock_nanosleep(), clock_nanosleep() sẽ cộng dồn thời gian ngủ từ điểm mốc khởi tạo, từ đây làm giảm sự liên quan đến chu trình xử lý lệnh còn nanosleep() sẽ ngủ khi chạy đến lệnh đó, việc tốn thời gian xử lý các lệnh trước nanosleep làm sai lệch thời gian lấy mẫu của bài. Bên cạnh đó sử dụng thêm khoá Mutex.

1.2. 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 *input_func(void *arg)
```

```
{
  for(int i = 0; ; i++)
  {
    pthread_mutex_lock(&mutex);
    FILE *f;
    f = fopen("freq.txt","r");

    int frequency_new = 0;
    fscanf(f,"%d", &frequency_new);
    fclose(f);

    if(frequency == frequency_new) pthread_mutex_unlock(&mutex);
    else
    {
      frequency = frequency_new;
      pthread_mutex_unlock(&mutex);
    }
  }
}
```

1.3. Thread LOGGING

Thread **LOGGING** chờ khi biến T được cập nhật mới, 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 *logging_func(void *arg)
{
  for(int i = 0; ; i++)
  {
    FILE *f;
    f = fopen("time_and_interval.txt","a+");
    long diff_sec = ((long) ts.tv_sec) - tt.tv_sec ;
    long diff_nsec;
```

```
    if(tt.tv_nsec != ts.tv_nsec || tt.tv_sec != ts.tv_sec)
    {
      if(ts.tv_nsec > tt.tv_nsec)
        {
          diff_nsec = ts.tv_nsec - tt.tv_nsec;
        }
      else
        {
          diff_nsec = MAX + ts.tv_nsec - tt.tv_nsec ;
          diff_sec = diff_sec - 1;
        }
       fprintf(f,"\n%ld.%09ld",diff_sec,diff_nsec);
       tt.tv_nsec =ts.tv_nsec;
       tt.tv_sec = ts.tv_sec;

    }
    fclose(f);
  }
}
```

2. Shell script để thay đổi giá trị chu kỳ X

Trong phần này, ta sẽ viết shell script để thay đổi lại giá trị chu kỳ X trong file "freq.txt" sau mỗi 1 phút. Và các giá trị X lần lượt được ghi như sau: 1000000 ns, 100000 ns, 10000 ns, 1000 ns, 100ns.

```
echo "1000000">freq.txt
sleep 10

echo "100000">freq.txt
sleep 60
```

```
echo "10000">freq.txt
sleep 60

echo "1000">freq.txt
sleep 60

echo "100">freq.txt
sleep 60
```

3. Thực hiện chạy shell script + chương trình C

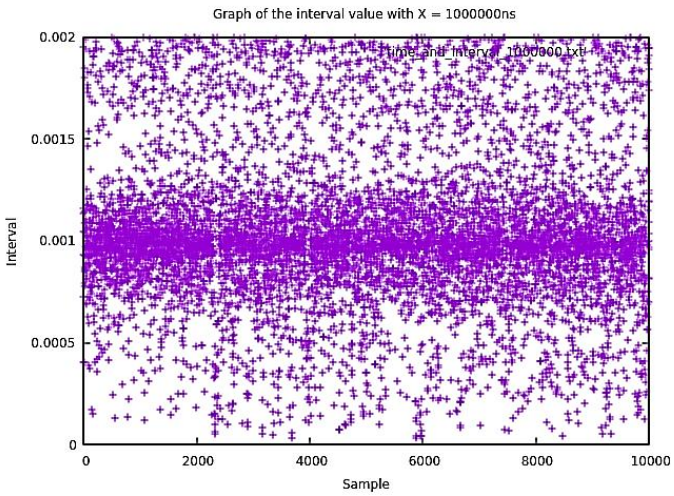
Trong phần này, ta sẽ cho chạy chương trình C và shell script trong vòng 5 phút, và sau đó dừng chương trình C.

4. Khảo sát giá trị interval

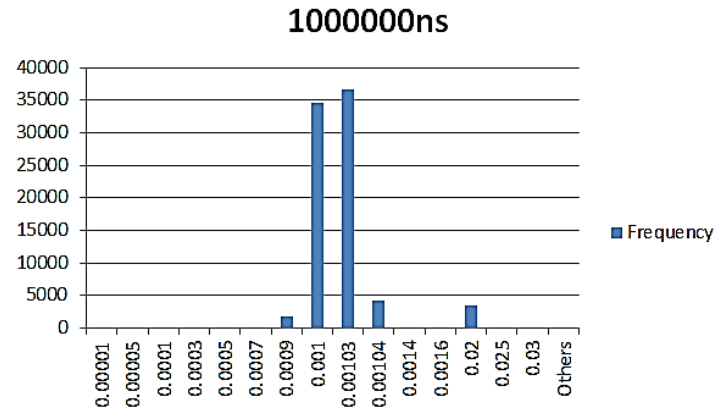
Trong phần cuối này, ta thực hiện khảo sát file "time_and_interval.txt".Bằng cách vẽ đồ thị giá trị và biểu đồ histogram của interval đối với mỗi giá trị chu kỳ X đồng thời đánh giá và đưa ra kết luận.

4.1. Với chu kỳ X = 1000000ns

Đồ thị giá trị và biểu đồ histogram của interval tương ứng với trường hợp chu kỳ X = 1000000ns được mô tả hình 4.1 và 4.2 dưới đây:



Hình 4. 1 Đồ thị giá trị interval khi chu kỳ X = 1000000ns

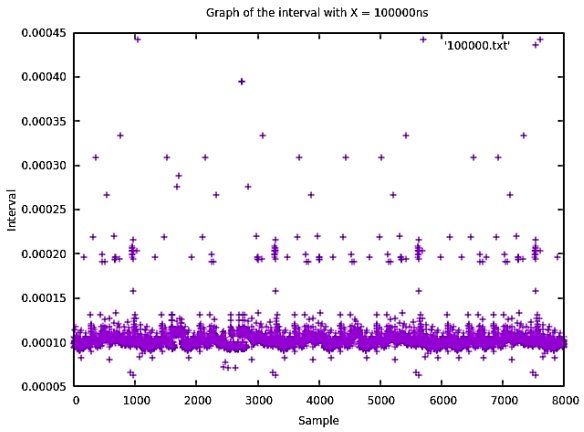


Hình 4. 2 Biểu đồ histogram của interval khi chu kỳ X = 1000000ns

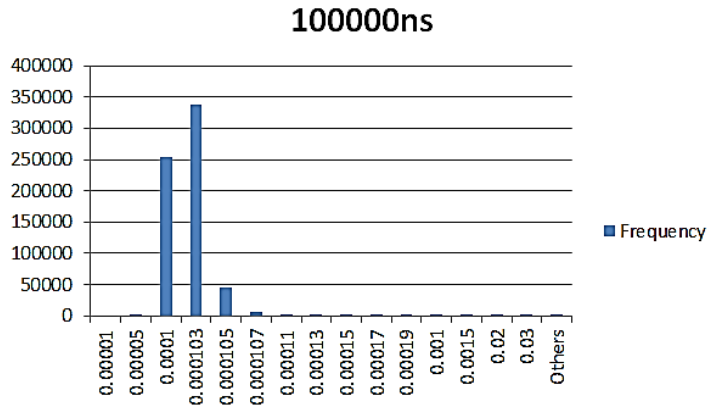
Dựa vào hình 4.1có thể thấy cụ thể hơn là giá trị interval dao động chủ yếu trong khoảng [0.0009; 0.00103s].

4.2. Với chu kỳ X = 100000ns

Quan sát lần lượt các hình 4.3 và 4.4 .dưới đây tương ứngvới đồ thị giá trị và biểu đồ histogram của interval tương ứng với trường hợp chu kỳ X = 100000ns.



Hình 4. 3 Đồ thị giá trị interval khi chu kỳ X = 100000ns

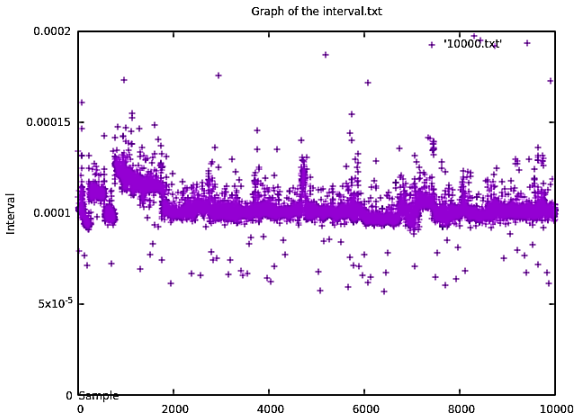


Hình 4. 4 Biểu đồ histogram của interval khi chu kỳ X = 1000000ns

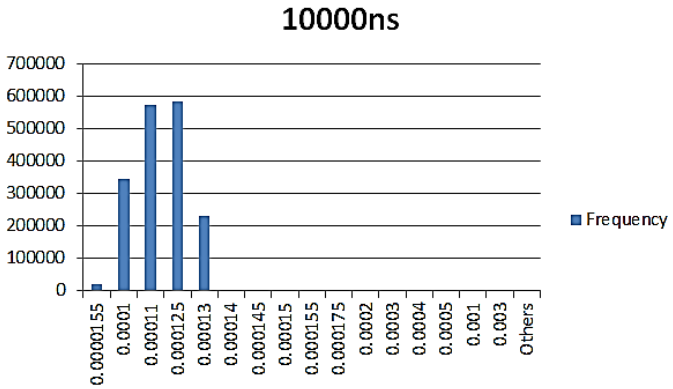
Tương tự như phần trên, tần suất của giá trị interval = 0.0001s là nhiều nhất. Dựa vào hình 4.2. có thể thấy cụ thể hơn là giá trị interval dao động chủ yếu trong khoảng [0.0001; 0.00105s]. Cải thiện rõ rệt so với trường hợp trước.

4.3. Với chu kỳ X = 10000ns

Đồ thị giá trị và biểu đồ histogram của interval tương ứng với trường hợp chu kỳ X = 10000ns được biểu diễn theo hình 4.5 và 4.6 dưới đây



Hình 4. 5 Đồ thị giá trị interval khi chu kỳ X = 10000ns

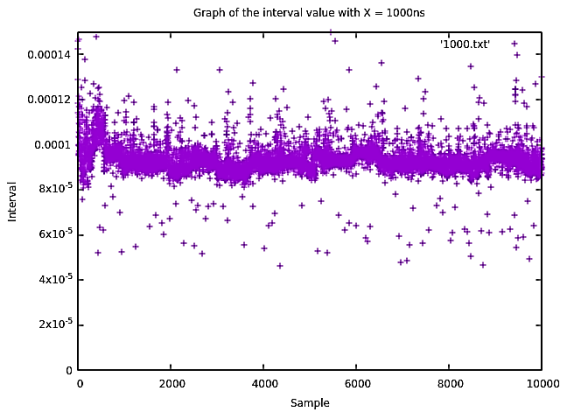


Hình 4. 6 Biểu đồ histogram của interval khi chu kỳ X = 10000ns

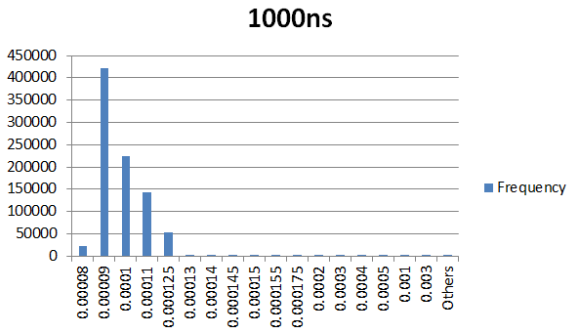
Từ hình 4.5, ta thấy tần suất của giá trị interval = 0.0001s là nhiều nhất. Giá trị interval dao động chủ yếu trong khoảng [0.0001;0.00013].

4.4. Với chu kỳ X = 1000ns

Ta có thể quan sát hình 4.7 và hình 4.8 dưới đây là đồ thị giá trị và biểu đồ histogram của interval tương ứng với trường hợp chu kỳ X = 1000ns



Hình 4. 7 Đồ thị giá trị interval khi chu kỳ X = 1000ns

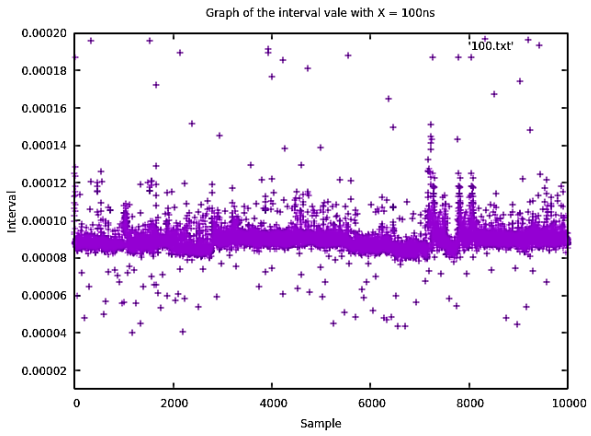


Hình 4. 8 Biểu đồ histogram của interval khi chu kỳ X = 1000ns

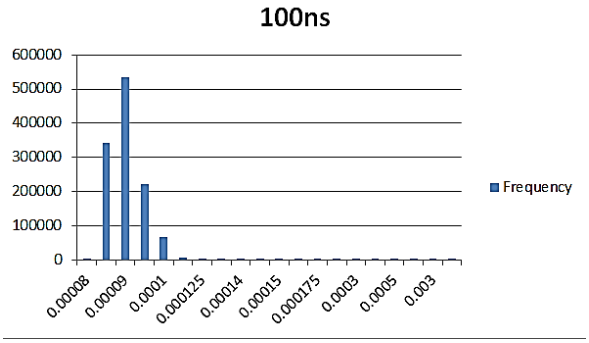
Từ hình 4.7. có thể thấy sự xuất hiện của giá trị interval = 0.00009s là nhiều nhất. Giá trị interval dao động chủ yếu trong khoảng [0.00008; 0.000125s].

4.5. Với chu kỳ X = 100ns

Đồ thị giá trị và biểu đồ histogram của interval tương ứng với trường hợp chu kỳ X = 100ns được mô tả lần lượt theo hình 4.9 và 4.10. dưới đây:



Hình 4. 9 Đồ thị giá trị interval khi chu kỳ X = 100ns



Hình 4. 10 Biểu đồ histogram của interval khi chu kỳ X = 100ns

Tương tự như phần trên, giá trị interval dao động chủ yếu trong khoảng [0.00085; 0.0001]s.

Kết luận: Khi chu kỳ càng lớn thì giá trị interval giữa T hiện tại và T của lần ghi trước càng lớn. Trong một vài trường hợp có xuất hiện giá trị thời gian lớn.

III. KẾT LUẬN

- Tiến độ công việc: Đã hoàn thành
- Khó khăn gặp phải:

- Kiến thức về lập trình C trên Linux và multithread, viết shell script còn khá mới mẻ;
- Thời gian tìm hiểu còn kéo dài: Phần lớn thời gian được sử dụng cho việc tham khảo về lập trình và multithread trên Internet;
- Chưa tối ưu được thuật toán một cách triệt để: Giá trị interval thu được chưa đạt như mong muốn;
- Thời gian debug lâu.