ĐẠI HỌC CÔNG NGHỆ - ĐHQGHN

KHOA ĐIỆN TỬ VIỄN THÔNG

----- ☙ 🕮 ❧ -----

Thực tập thiết kế hệ thống nhúng

Nhóm thực tập Nhúng- Dự án gNode 5G VHT

Báo cáo tuần 1

LINUX PROGRAMMING ASSIGMENT

Sinh viên thực hiện:

Đỗ Thái Vũ - 19021540

Hà Nội, ngày 15 tháng 7 năm 2022

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

1.1 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**.

```
void *getTime(void *args )
{
    clock_gettime(CLOCK_REALTIME,&request1);
  while(check_loop == 1)
  {
    clock_gettime(CLOCK_REALTIME,&tp);
    long temp;
      if(request1.tv_nsec + freq > 1000000000)
      {
        temp = request1.tv_nsec;
        request1.tv_nsec = temp + freq - 1000000000;;
        request1.tv_sec +=1;
            if(clock_nanosleep(CLOCK_REALTIME,TIMER_ABSTIME, &request1,NULL) != 0)
            {
                check_loop = 0;
            }
            else
            {
                check_loop = 1;
            }

      }
      else{
          request1.tv_nsec +=freq;
```

```
        if(clock_nanosleep(CLOCK_REALTIME,TIMER_ABSTIME, &request1,NULL) != 0)
         {
            check_loop = 0;
         }
        else
         {
            check_loop = 1;
         }
      }
  }
}
```

Lưu ý :

- Khi tín hiệu được phân phối ở tốc độ cao thì nanosleep có thể dẫn đến sự thiếu chính xác lớn trong cả 1 quy trình (do thời gian sleep), để giải quyết vấn đề, lần call đầu tiên dùng hàm clock_gettime() để truy xuất thời gian, sau đó thêm thời lượng mong muốn vào. Cuối cùng gọi clock_nanosleep() với TIMER_ABSTIME flag.

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 *getFreq(void *args)
{
  while(1)
  {
  pthread_mutex_lock(&mutex);
  FILE *fp;
  fp = fopen("freq.txt","r");
   unsigned long new_freq = 0;
  fscanf(fp,"%lu",&new_freq);
  fclose(fp);
  if(new_freq == freq)
  {  pthread_mutex_unlock(&mutex);
  }
    else
  {
    freq = new_freq;
    pthread_mutex_unlock(&mutex);
  }
  }
}
```

1.3. 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)
{
  while(1)
  {
    FILE *file;
    file = fopen("interval.txt","a+");
    long diff_sec = ((long) tp.tv_sec) - tmp.tv_sec ;
    long diff_nsec;
    if(tmp.tv_nsec != tp.tv_nsec || tmp.tv_sec != tp.tv_sec)
    {
      if(tp.tv_nsec > tmp.tv_nsec)
        {
          diff_nsec = tp.tv_nsec - tmp.tv_nsec;
        }
      else
        {
          diff_nsec = 1000000000 + tp.tv_nsec - tmp.tv_nsec ;
          diff_sec = diff_sec - 1;
        }
      fprintf(file,"\n%ld.%09ld",diff_sec,diff_nsec);
      tmp.tv_nsec =tp.tv_nsec;
      tmp.tv_sec = tp.tv_sec;
    }
    fclose(file);
  }
```

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

File script.sh

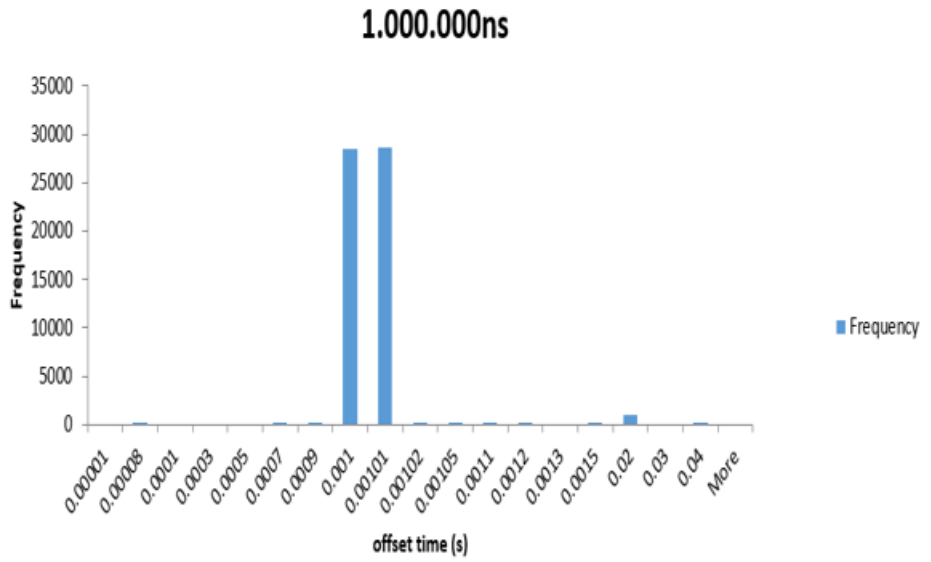
```
#!/bin/sh
echo "1000000">freq.txt
timeout 60s ./b1
echo "100000">freq.txt
timeout 60s ./b1
echo "10000">freq.txt
timeout 60s ./b1
echo "1000">freq.txt
timeout 60s ./b1
echo "100">freq.txt
timeout 60s ./b1
```

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 (theo lệnh trên thì chương trình mỗi freq chỉ chạy trong 60 s).

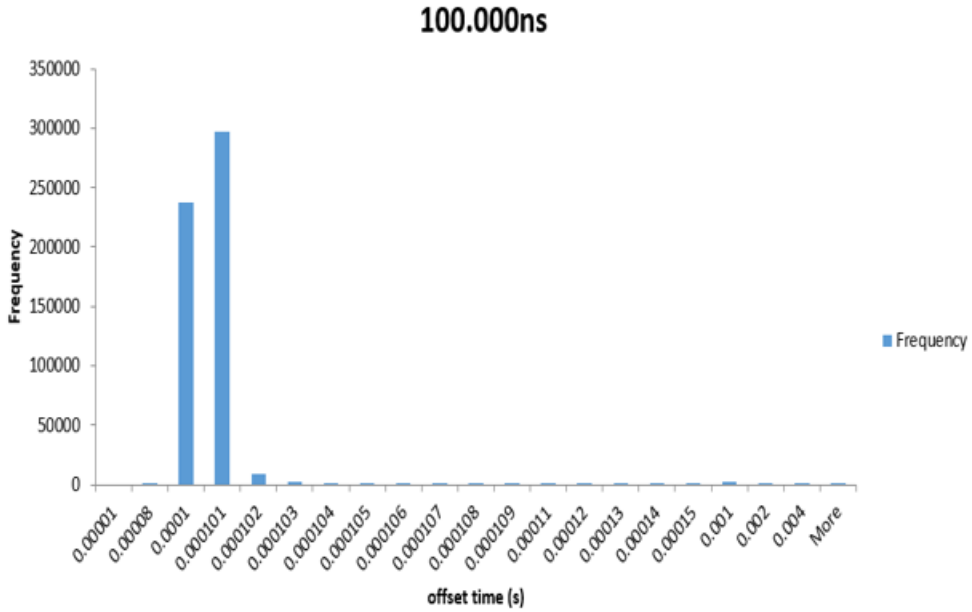
4. Khảo sát giá trị interval

4.1. Với chu kỳ X = 1000000ns



Hình 4.1 Biểu đồ histogram khi chu kỳ X = 1000000ns

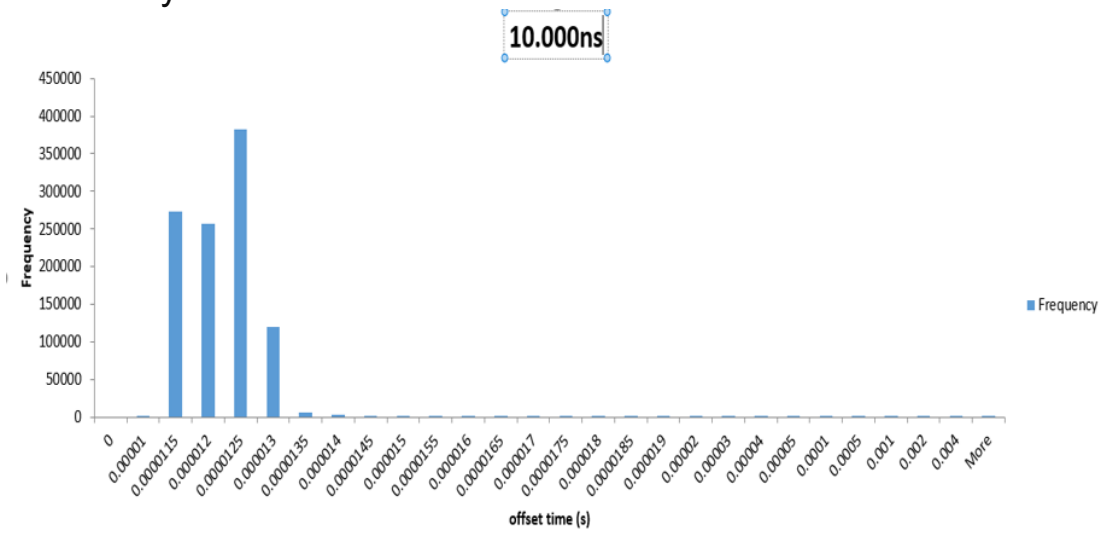
Nhận xét : ta có thể thấy các interval chủ yếu ở 0.001s và 0.00101s, chủ yếu trong khoảng [0.001-0.002].4.2. Với chu kỳ X = 100000ns



Hình 4.2 Biểu đồ histogram khi chu kỳ X = 100000ns

Nhận xét : Các interval xuất hiện nhiều ở offset 0.0001s và 0.000101s , chủ yếu nằm trong khoảng [0.0001-0.00012]

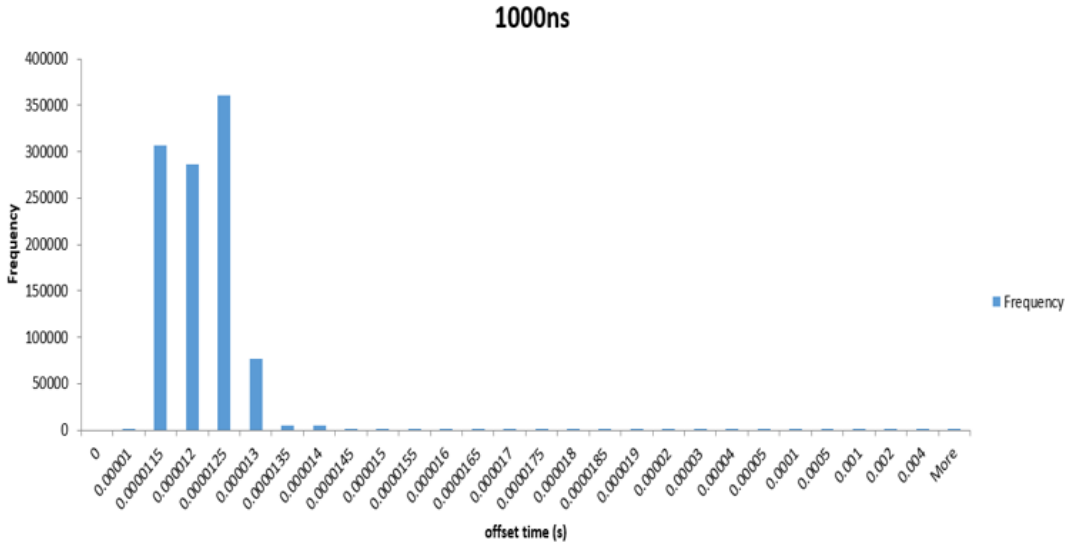
4.3. Với chu kỳ X = 10000ns



Hình 4.3 Biểu đồ histogram khi chu kỳ X = 10000ns

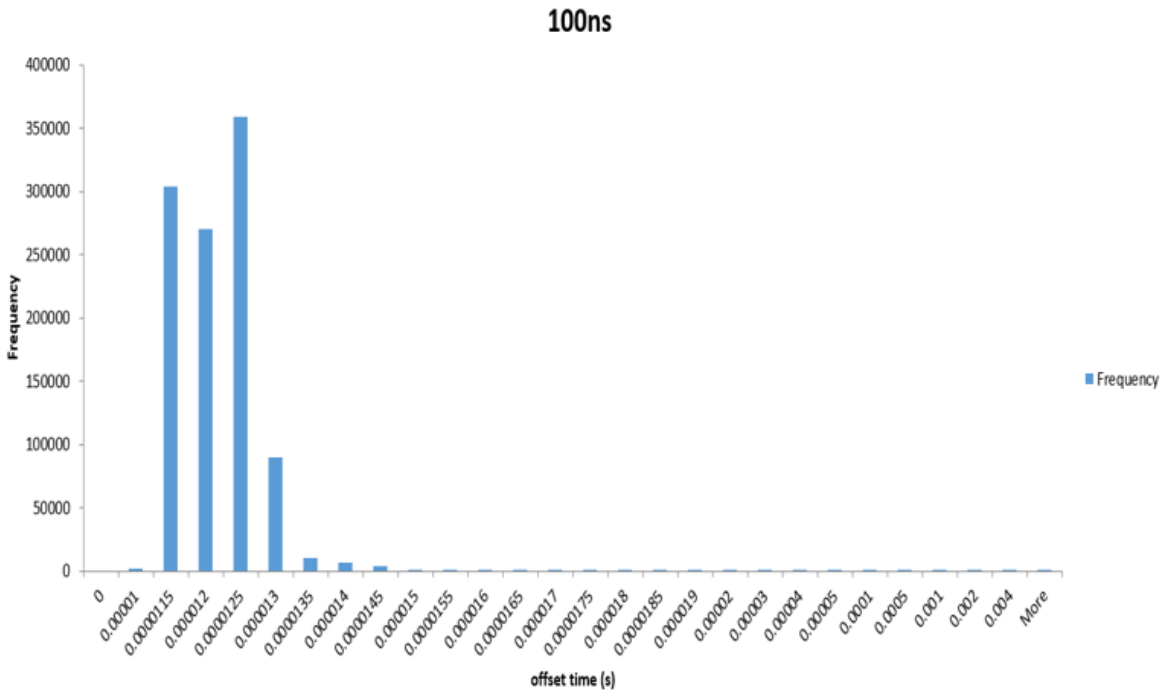
Nhận xét: Các Interval xuất hiện nhiều ở offset 0.0000125s, 0.000012, 0.00001s

4.4. Với chu kỳ X = 1000ns



Hình 4.4 Biểu đồ histogram khi chu kỳ X = 1000ns

4.5. Với chu kỳ X = 100ns



Hình 4.5 Biểu đồ histogram khi chu kỳ X = 100ns

Nhận xét chung :

- Khi chu kỳ càng lớn thì giá trị interval giữa T hiện tại và T của lần ghi trước càng lớn.

- Trong các lần lấy mẫu đều xuất hiện thời gian lớn (0.003s)

5. Kết luận

- Tiến độ công việc: Đã hoàn thành trước deadline (15/7/2022)

- Khó khăn gặp phải :

+Shell Script trên Hệ ĐH Linux

+ Kiến thức về C như Threading , Set time

+ Chưa tối ưu được thuật toán một cách tối ưu: Giá trị interval thu được chưa đạt như mong muốn;

6. Lý thuyết bổ sung

a. Multi Threading

Giống như các quy trình, các luồng là một cơ chế cho phép ứng dụng thực hiện nhiều nhiệm vụ đồng thời. Một quy trình duy nhất có thể chứa nhiều luồng. Tất cả các luồng này đều thực thi độc lập giống nhau và tất cả chúng đều chia sẻ cùng một bộ nhớ chung, bao gồm cả dữ liệu được khởi tạo, dữ liệu chưa khởi tạo và phân đoạn đống. (Quy trình UNIX truyền thống chỉ đơn giản là một quy trình đặc biệt trường hợp của một quy trình đa luồng; nó là một quá trình chỉ chứa một chuỗi.)

- Chia tiến trình dùng hàm fork() nhưng có nhược điểm:

+ Khó để chia sẻ thông tin giữa các processes. Bởi vì giữa parent and child ko share memory

+ Quá trình tạo fork() tương đối tốn kém

-Threads có thể giải quyết dc vấn đề đó:

+Có thể chia sẻ thông tin giữa các thread (luồng) nhanh và dễ.

+Thread creation nhanh hơn process creation-typically 10 lần or better.

- Bên cạnh global memory (bộ nhớ chung), thread còn share 1 số thuộc tính chung cho 1 process, thay vì cụ thể cho 1 thread).

Các thuộc tính này bao gồm (663 pdf)

-Background Details of the Pthreads API

Data type	Description
pthread_t	Thread identifier
pthread_mutex_t	Mutex
pthread_mutexattr_t	Mutex attributes object
pthread_cond_t	Condition variable
pthread_condattr_t	Condition variable attributes object
pthread_key_t	Key for thread-specific data
pthread_once_t	One-time initialization control context
pthread_attr_t	Thread attributes object

- Các lệnh thường được dung là :

```
+ pthread_create(&thread,NULL,func,&arg)
+ pthread_exit(void *retval)
+ pthread_cancel()
+ pthread_self(void)
+ pthread_join()
```

b. Mutex

+ Mutex được hiểu là đối tượng (hoặc cờ hiệu) mang 2 trạng thái là đang được sử dụng và chưa được sử dụng (trạng thái sẵn sàng)

+ Khai báo mutex cấp tốc :

```
Pthread_mutex_t a_mutex = PTHREAD_MUTEX_INITIALIZER;
```

Lưu ý : Đối tượng mutex này không thể bị khóa 2 lần bởi cùng 1 luồng. Trong luồng nếu bạn đã gọi hàm khóa mutex này và thực hiện khóa mutex 1 lần nữa, bạn sẽ rơi vào trạng thái khóa chết (deadlock).

Có thể dùng

```
pthread_mutex_init(pthread_mutex_t * mutex, const pthread_mutexattr_t * mutexattr);
```

+ Khóa và tháo khóa

```
int rc = pthread_mutex _lock( & a_mutex);
```

rc = 1 thì có lỗi ; rc = 0 thì khóa lại thành công.

Mở khóa để trả lại tài nguyên cho luồng khác

```
int rc = pthread_mutex _unlock( & a_mutex);
```

rc = 1 thì có lỗi ; rc = 0 thì mở khóa lại thành công.

+ Hủy mutex (nên hủy sau khi dùng xong)

```
rc = pthrea_mutex_destroy (& a_mutex)
```

c. Timer and sleep

- Các lệnh gọi hệ thống chính trong API đồng hồ POSIX là:

+ clock_gettime () :lấy giá trị hiện tại của đồng hồ;

+ clock_getres () : trả về độ phân giải của đồng hồ;

+clock_settime () : cập nhật đồng hồ.

-Hàm nanosleep() để tạo 1 thời gian “ngủ” cho chương trình. Cần sử dụng thư viện unistd.h

- Khi tín hiệu được phân phối ở tốc độ cao thì nanosleep có thể dẫn đến sự thiếu chính xác lớn trong cả 1 quy trình (do thời gian sleep), để giải quyết vấn đề, lần call đầu tiên dùng hàm clock_gettime() để truy xuất thời gian, sau đó thêm thời lượng mong muốn vào. Cuối cùng gọi clock_nanosleep() với TIMER_ABSTIME flag.

- Một số struct quan trọng :

```
struct timespec {
 time_t tv_sec; /* Seconds ('time_t' is an integer type) */
 long tv_nsec; /* Nanoseconds */
};
```

```
struct timespec request;
 /* Retrieve current value of CLOCK_REALTIME clock Page 538 */
 if (clock_gettime(CLOCK_REALTIME, &request) == -1)
 errExit("clock_gettime");
 request.tv_sec += 20; /* Sleep for 20 seconds from now */
 s = clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &request,
NULL);
 if (s != 0) {
 if (s == EINTR)
```

```
printf("Interrupted by signal handler\n");
else
errExitEN(s, "clock_nanosleep");
}
```