FACULTY OF INFORMATION TECHNOLOGY

>>> COURSE MATERIAL <<<

INTRODUCTION TO OPERATING SYSTEM

Course ID 502047

Lab Part 4 – Threads programming + File manipulation

Mục tiêu	**Lý thuyết liên quan**	**Tài nguyên**
Tạo thread	Ch4: Threads and Concurrency.	Sử dụng image Ubuntu 14 / 16
Kết thúc thread		
Đồng bộ		
Thao tác tập tin	SV tự học	
Programming Problems	Ch2: OS structure	Lab3_source.rar
Programming Problems	Ch3: Process	

Yêu cầu nộp bài: các tập tin mã nguồn .c và tập tin khả thực thi .out của các "ví dụ" và bài tập cuối hướng dẫn.

Preferences

[1] Abraham Silberschatz, Peter B. Galvin, Greg Gagne, [2018], Operating System Concepts, 10th edition, John Wiley & Sons, New Jersey.

Programming Problems of Chapter 3.

[2] Nguyễn Hồng Vũ, [2016] Hướng dẫn LAB Hệ điều hành, khoa Công nghệ thông tin, ĐH Tôn Đức Thắng.

[3] Greg Gagne , [2019], GitHub OS-BOOK OSC10e, Westminster College, United States

Access https://github.com/greggagne/osc10e in September 2019.

[4] Basics of File Handling in C, *geeksforgeeks.org*

1. Tạo threads

- Thư viện hàm sử dụng:

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

- Hàm khởi tạo thread

```
int pthread_create(pthread_t *thread, pthread_attr_t *attr, void
*(*start_routine)(void*) *arg)
```

- Kết quả trả về

- 0: Thành công, tạo ra một thread mới, ID của thread được trả về qua đối số thread
- <>0: thất bại

- Đối số truyền:

- thread: dùng để giữ tham khảo đến threadID nếu hàm thành công. Kiểu pthread_t
- att: giữ thuộc tính của thread, set NULL nếu muốn sử dụng các thuộc tính mặc định của hệ thống.

```
typedef struct __pthread_attr_s {
    int __detachstate;
    int __schedpolicy;
    struct __sched_param __schedparam;
    int __inheritsched;
    int __scope;
    size_t __guardsize;
    int __stackaddr_set;
    void *__stackaddr;
    size_t __stacksize;
} pthread_attr_t;
```

- start_routine: là một hàm do người sử dụng định nghĩa mà sẽ được thực thi bởi thread mới.

- Hàm này nên có kiểu trả về là một con trỏ kiểu void, nếu không thì phải ép kiểu về con trỏ void khi gọi thread.
- Hàm này nên có một đối số truyền cũng có kiểu con trỏ void.
- Đối số truyền của hàm này sẽ được truyền thông qua tham số thứ 4 (arg)
- arg: là đối số truyền cho hàm start_routine
- Trong trường hợp đối số truyền cần nhiều hơn 1 tham số có thể sử dụng struct, các đối số kiểu này phải được cấp phát tĩnh và được khởi tạo trước.

- Khi thread được tạo ra, nó có những thuộc tính riêng, một stack thực thi. Nó kế thừa các tín hiệu và độ ưu tiên từ chương trình gọi.

2. Kết thúc threads

- Khi thread được tạo ra, nó thực thi hàm start_routine cho đến khi hoàn tất.

- Có lời gọi hàm thread_exit tường minh

- Bị cancel bởi hàm thread_cancel

- Tiến trình tạo ra thread kết thúc

- Có một thread gọi system call exec

Ví dụ 1: Hãy biên dịch, chạy thử, bỏ hàm sleep() ở dòng 17, hoặc tăng giá trị đối số sleep() và giải thích kết quả chạy cho từng tình huống.

```
1       #include <pthread.h>
2       #include <stdio.h>
3
4       void* thr1(void* ar){
5           printf("This is thread %d\n",*((int*) ar));
6           sleep(2);
7       }
8
9       int main(int argc, char* argv[]){
10          int i;
11          int num=atoi(argv[1]);
12          pthread_t tid[num];
13          for(i=0; i<num; i++){
14*             pthread_create(&tid[i], NULL, thr1, (void*)
15      &tid[i]);
16          }
17*         sleep(3);
18          return(0);
        }
```

```
$ gcc -o t01.out t01.o -lpthread
$ ./t01.out 3
This is thread 1547249408
This is thread 1555654312
This is thread 1564399124
```

Ví dụ 2:

- Truyền dữ liệu cho thread, tất cả những gì cần truyền đặt vào trong một struct.

- Đối số thứ 4 là một kiểu struct

- Lưu ý phần ép kiểu trong thr1

- Lưu ý đối số truyền thứ 4 của hàm pthread_create

```
1       #include <pthread.h>
2       #include <stdio.h>
3       #include <stdlib.h>
4       #include <sys/types.h>
5       #include <unistd.h>
6       struct arr{
7            int n;
8            int a[10];
9       };
10      void* thr1(void* ar){
11           int count;
12           struct arr *ap=(struct arr*) ar;
13           for(count=0;count<ap->n; count++)
14*               printf("%d\t",ap->a[count]);
15           printf("\n");
16      }
17
18      int main(int argc, char* argv[]){
19           struct arr ar;
20           ar.n=5;
21           int i;
22           for(i=0;i<ar.n;i++)
23                ar.a[i]=i+1;
24           pthread_t tid;
25           pthread_create(&tid,NULL,&thr1,&ar);
26           sleep(2);
```

```
27      return 0;
28  }
```

```
$ gcc -o t02.out t02.o -lpthread
$ ./t02.out
1     2     3     4     5
```

- Đợi tiểu trình hoàn tất

- Tiến trình gọi có thể sử dụng lệnh pthread_join để đợi tiểu trình con hoàn tất.

```
int pthread_join(pthread_t threadid, void **status)
```

- Kết quả trả về:
  - Thành công: 0
  - Thất bại: <>0
- Đối số:
  - threadid: một threadid đã tồn tại
  - status: trạng thái trả về, 0 nếu thành công.

Ví dụ 3:

- Sleep() là một giải pháp không tốt, ngay cả khi hàm gọi sleep() dừng lại trong vài giây vẫn không đảm bảo rằng các tiểu trình con bắt theo kịp, điều đó phụ thuộc vào bộ lập lịch của CPU. Hơn nữa, xác định một tham số k cho sleep(k) là khó khăn. Nếu k quá lớn, tiến trình phải chờ lãng phí, nếu k quá bé, hàm main() không đợi kịp tiểu trình con.

`pthread_join` là lời gọi để tiến trình `main()` hoặc một tiểu trình chờ tín hiệu kết thúc của một tiểu trình khác, được chỉ định thông qua tham số.

```
1   #include <pthread.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include <sys/types.h>
5   #include <unistd.h>
6   void* thr1(void* ar){
7       int count;
8       printf("This is thread %d\n",*((int*)ar));
9       sleep(2);
10  }
11  int main (int argc, char* argv[]){
12      int i;
```

```
13      pthread_t tid[3];
14*     int status, *pstatus = &status;
15      for(i=0;i<3;i++)
16           pthread_create(&tid[i],NULL,thr1,(void*)
17  &tid[i]);
18      for(i=0;i<3;i++){
19           if(pthread_join(tid[i],(void**) pstatus)>0){
20                printf("pthread_join for thread %d
21  failure\n", (int)tid[i]);
22           }
23      printf("pthread_waited of %d OK, return code:
24  %d\n", (int)tid[i], status);
25      sleep(1);
26      }
27      sleep(1);
28  return 0;
29  }
```

```
$ gcc -o t03.out t03.o -lpthread
$ ./t03.out
This is thread 1547249408
This is thread 1555654312
This is thread 1564399124
pthread_waited of -1547249408 OK, return code: 0
pthread_waited of -1547249408 OK, return code: 0
pthread_joid for thread -1555654312 failure
```

Ví dụ 4: tổng hợp các ví dụ đã nêu.
- Hàm thr1 sử dụng để khởi tạo mảng
- Hàm thr2 sử dụng để tính tổng các phần tử của mảng.
- Hàm thr3 sử dụng để ghi mảng ra file

Lưu ý việc sử dụng hàm pthread_join.

```
1   #include <pthread.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include <sys/types.h>
5   #include <unistd.h>
6   struct arr{
7        int n;
8        int a[10];
```

```
9       };
10      struct file {
11           struct arr ar;
12           char* filename;
13      };
14*     static int sum =0;
15      void* thr1(void* ar){
16           struct arr *ap = (struct arr*) ar;
17           ap->n=3; int i=0;
18           for(i=0;i<ap->n;i++)
19                ap->a[i] = i+1;
20      }
21
22      void* thr2(void* ar){
23           struct arr *ap = (struct arr*) ar;
24           int i, s=0;
25           for(i=0;i<ap->n;i++)
26                s=s + ap->a[i];
27           sum=s;
28
29      }
30
31      void* thr3 (void* ar){
32           struct file *fi = (struct file*) ar;
33           FILE *out; int count;
34           out= fopen(fi->filename,"wb");
35           fprintf(out,"number element or array: %d\n", fi-
36      >ar.n);
37           for(count=0; count<fi->ar.n; count++){
38                fprintf(out,"%d\t",fi->ar.a[count]);
39           }
40           fprintf(out,"\n");
41           fprintf(out,"sum=%d\n",sum);
42           printf("tong : %d\n",sum);
43           fclose(out);
44      }

M1      int main (int argc,char * argv[]){
02           int i;
03           pthread_t tid[3];
04           struct arr ar;
05           int status, *pstatus= &status;
06           pthread_create(&tid[0],NULL,thr1,(void*) &ar);
07           sleep(1);
08           if(pthread_join(tid[0],(void**) pstatus)==0)
09           {
```

```
10          pthread_create(&tid[1],NULL,thr2,(void*) &ar);
11          if(pthread_create(&tid[1],NULL,thr2, (void*)
12    &ar)==0)
13          {
14               struct file arf;
15               arf.ar=ar;
16               arf.filename=argv[1];
17               pthread_create(&tid[2],NULL,thr3,(void*)
18    &arf);
19          }
20     }
21     sleep(2);
22     return 0;
23    }
```

```
$ gcc -o t04.out t04.o -lpthread
$ ./t04.out tf1
$ cat tf1
Number element of array: 3
1     2     3
sum = 6
```

3. Thao tác trên tập tin

(SV tự học)

1. Creation of a new file (**fopen with attributes as "a" or "a+" or "w" or "w++"**)
2. Opening an existing file (**fopen**)
3. Reading from file (**fscanf or fgetc**)
4. Writing to a file (**fprintf or fputs**)
5. Moving to a specific location in a file (**fseek, rewind**)
6. Closing a file (**fclose**)

Functions in File Operations:

File operation	Declaration & Description
fopen() – To open a file	Declaration: FILE *fopen (const char *filename, const char *mode) fopen() function is used to open a file to perform operations such as reading, writing etc. In a C program, we declare a file pointer and use fopen() as below. fopen() function creates a new file if the mentioned file name does not exist. FILE *fp; fp=fopen ("filename", "'mode"); Where, fp – file pointer to the data type "FILE". filename – the actual file name with full path of the file. mode – refers to the operation that will be performed on the file. Example: r, w, a, r+, w+ and a+. Please refer below the description for these mode of operations.
fclose() – To close a file	Declaration: int fclose(FILE *fp); fclose() function closes the file that is being pointed by file pointer fp. In a C program, we close a file as below. fclose (fp);
fgets() – To read a file	Declaration: char *fgets(char *string, int n, FILE *fp) fgets function is used to read a file line by line. In a C program, we use fgets function as below. fgets (buffer, size, fp); where, buffer – buffer to put the data in. size – size of the buffer fp – file pointer
fprintf() – To write into a file	Declaration: int fprintf(FILE *fp, const char *format, ...);fprintf() function writes string into a file pointed by fp. In a C program, we write string into a file as below. fprintf (fp, "some data"); or fprintf (fp, "text %d", variable_name);

Opening or creating file

For opening a file, fopen function is used with the required access modes. Some of the commonly used file access modes are mentioned below.

File opening modes in C:

- "r" – Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer which points to the first character in it. If the file cannot be opened fopen( ) returns NULL.

- “w” – Searches file. If the file exists, its contents are overwritten. If the file doesn’t exist, a new file is created. Returns NULL, if unable to open file.
- “a” – Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer that points to the last character in it. If the file doesn’t exist, a new file is created. Returns NULL, if unable to open file.
- “r+” – Searches file. If is opened successfully fopen( ) loads it into memory and sets up a pointer which points to the first character in it. Returns NULL, if unable to open the file.
- “w+” – Searches file. If the file exists, its contents are overwritten. If the file doesn’t exist a new file is created. Returns NULL, if unable to open file.
- “a+” – Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer which points to the last character in it. If the file doesn’t exist, a new file is created. Returns NULL, if unable to open file.

Writing a file –:

The file write operations can be perfomed by the functions fprintf and fputs with similarities to read operations. The snippet for writing to a file is as :

FILE *filePointer ;

filePointer = fopen(“fileName.txt”, “w”);

fprintf(filePointer, "%s %s %s %d", "We", "are", "in", 2019);

-

Closing a file –:

After every successful fie operations, you must always close a file. For closing a file, you have to use fclose function. The snippet for closing a file is given as :
FILE *filePointer ;

filePointer= fopen(“fileName.txt”, “w”);

---------- Some file Operations -------

- fclose(filePointer)

	`// C program to Open a File,` `// Write in it, And Close the File`

```
# include <stdio.h>
# include <string.h>

int main( )
{

    // Declare the file pointer
    FILE *filePointer ;

    // Get the data to be written in file
    char dataToBeWritten[50]
      = "GeeksforGeeks-A Computer Science Portal for
Geeks";

    // Open the existing file GfgTest.c using fopen()
    // in write mode using "w" attribute
    filePointer = fopen("GfgTest.c", "w") ;

    // Check if this filePointer is null
    // which maybe if the file does not exist
    if ( filePointer == NULL )
    {
      printf( "GfgTest.c file failed to open." ) ;
    }
    else
    {

      printf("The file is now opened.\n") ;

      // Write the dataToBeWritten into the file
      if ( strlen ( dataToBeWritten ) > 0 )
      {

           // writing in the file using fputs()
           fputs(dataToBeWritten, filePointer) ;
           fputs("\n", filePointer) ;
      }

      // Closing the file using fclose()
      fclose(filePointer) ;

      printf("Data successfully written in file
GfgTest.c\n");
      printf("The file is now closed.") ;
    }
    return 0;
```

```
}
```

```
// C program to Open a File,
// Read from it, And Close the File

# include <stdio.h>
# include <string.h>

int main( )
{

    // Declare the file pointer
    FILE *filePointer ;

    // Declare the variable for the data to be read
from file
    char dataToBeRead[50];

    // Open the existing file GfgTest.c using fopen()
    // in read mode using "r" attribute
    filePointer = fopen("GfgTest.c", "r") ;

    // Check if this filePointer is null
    // which maybe if the file does not exist
    if ( filePointer == NULL )
    {
      printf( "GfgTest.c file failed to open." ) ;
    }
    else
    {

      printf("The file is now opened.\n") ;

      // Read the dataToBeRead from the file
      // using fgets() method
      while( fgets ( dataToBeRead, 50, filePointer ) !=
NULL )
      {

           // Print the dataToBeRead
           printf( "%s" , dataToBeRead ) ;
      }

      // Closing the file using fclose()
```

```
      fclose(filePointer) ;

      printf("Data successfully read from file
GfgTest.c\n");
      printf("The file is now closed.") ;
    }
    return 0;
}
```

Vấn đề lập trình

Xin xem file "Exercise 04.pdf" và thảo luận / trả lời các câu hỏi 4.17, 4.19, trong file đó, 4.22 trở đi.

Bài tập lập trình

1. (Bài tập 4.22) Viết chương trình đa luồng tính toán các giá trị thống kê khác nhau từ một danh sách các số được truyền vào thông qua đối số của dòng lệnh. Chương trình sau đó sẽ tạo ba tiểu trình tính toán riêng biệt. Một tiểu trình sẽ xác định trung bình cộng của các số, tiểu trình thứ hai sẽ xác định giá trị lớn nhất và tiểu trình thứ ba sẽ xác định giá trị nhỏ nhất. Ví dụ:

```
>./bai22.out 90 81 78 95 79 72 85
 Gia tri trung binh: 82
 Gia tri lon nhat: 95
 Gia tri nho nhat: 72
```

Các biến số đại diện cho các giá trị trung bình, nhỏ nhất và lớn nhất sẽ được lưu trữ trên toàn cục. Các tiểu trình sẽ tính toán các giá trị này và tiến trình cha sẽ xuất ra các giá trị kết quả khi tiểu trình kết thúc.

Mở rộng: có thể tạo thêm các tiểu trình để tính giá trị trung vị; độ lệch chuẩn; sắp xếp dãy số, ...

2. (Bài tập 4.23) Viết chương trình đa luồng để xuất ra số nguyên tố. Người dùng chạy chương trình và nhập vào một số nguyên thông qua đối số tại dòng lệnh. Chương trình sau

đó sẽ tạo ra một tiến trình riêng biệt xuất ra tất cả các số nguyên tố nhỏ hơn hoặc bằng số được nhập bởi người dùng.

3. Chỉnh sửa lại bài 1 sao cho thay vì xuất kết quả ra màn hình thì kết quả sẽ được ghi vào tập tin result.txt ở cùng thư mục với chương trình chạy.

4. Viết chương trình để sao chép dữ liệu từ file nguồn vào file đích. Với tên file nguồn và file đích là đối số đầu vào. Đếm số kí tự đã sao chép. Giả sử chương trình chỉ chạy trên tập tin text.

```
>./baitap4.out source.txt target.txt
 Da sao chep thanh cong 129 ki tu.
```

5. Viết chương trình tạo ra 3 thread thực hiện các công việc sau:

- Thread thứ nhất nhận đối số truyền từ môi trường (argv[1]), kiểm tra nếu số này lớn 0, tính giai thừa của số này (có thể sử dụng struct hoặc biến toàn cục).
- Thread thứ hai chờ thread thứ nhất hoàn tất, thực hiện việc tính tổng các số chẵn từ nhỏ hơn kết quả giai thừa thread thứ nhất tính được.
- Thread thứ ba chờ thread thứ 2 hoàn tất, ghi kết quả vào file là đối số thứ 2 từ biến môi trường (argv[2]). Nội dung ghi vào gồm: dòng thứ nhất lưu giá trị của argv[1]; dòng thứ 2 lưu kết quả của giai thừa; dòng thứ 3 lưu tổng các số chẵn nhỏ hơn giai thừa.

```
>./baitap5.out 4 result.txt
 Da tao ra tap tin result.txt
```

Và nội dung tập tin result.txt nên là

n = 4

4! = 24

sum = 156

6. Cho một tập tin có cấu trúc sau:

- Dòng đầu tiên chứa số phần tử mảng

- Dòng còn lại chứa các phần tử là số nguyên

Viết chương trình gồm các thread thực hiện các công việc sau:

- Thread thứ nhất đọc file đầu vào là đối số thứ nhất từ biến môi trường
- Thread thứ hai tính tổng các số nguyên tố trong mảng
- Thread thứ ba tính sắp xếp mảng tăng dần
- Thread thứ tư thực hiện việc ghi file result. Nội dung file đầu vào và đầu ra như sau:

Ví dụ: file đầu vào có dạng sau

```
10
4 5 7 8 11 9 20 13 2 3
```

Kết quả trong file result có dạng sau:

```
So phan tu mang: 10
4 5 7 8 11 9 20 13 2 3
Mang cac so nguyen to:
5 7 11 13 2 3
Tong cac so nguyen to: 41
Mang cac so nguyen to da duoc sap xep
2 3 5 7 11 13
```