实验7：线程并发

Pthreads

- Pthreads是POSIX的线程创建和同步的接口标准
- Pthreads定义了线程的行为，并未规定具体的实现方式
- 许多操作系统都实现了Pthreads标准
  - Solaris
  - Linux
  - Mac OS X
  - Tru64 UNIX

创建线程

- int pthread_create(pthread_t *tid, const pthread_attr_t *attr, void *(*start_rtn)(void), void *arg)
  - tid: 要创建的线程的线程id指针
  - attr: 创建线程时的线程属性，如不需要，可设为NULL；
  - start_rtn：返回值是void类型的指针函数，是新线程执行的函数
  - arg: 传递给start_rtn的参数
  - 返回值为0表示创建成功，非0表示失败

线程等待

- int pthread_join(pthread_t thread, void **status);
  - thread：指定要等待的线程的线程标识符
  - status: 用于存放线程采用pthread_exit()返回时的返回值
  - 返回0表示函数执行成功，非0表示失败

线程运行结束

- 两种方式结束
  - 自动返回，由return((void*)0)或return 0;
  - 通过pthread_exit()返回

    void pthread_exit(void *rval_ptr);
    - rval_ptr线程退出返回值的指针，可以由pthread_join函数中的第二个参数捕获

```
#include <pthread.h>
#include <stdio.h>
int sum;    /* this data is shared by the threads */
void* runner(void* param);  /* the thread */
int main(int argc, char *argv[])
{
   pthread_t tid ;    /* the thread identifier; */
   pthread_attr_t attr;  /* set of thread attributes  */
   if(argc!=2){
     fprintf(stderr, "Usage: %s <integer value>\n", argv[0]);
     return -1;
   }
   if(atoi(argv[1]) < 0){
     fprintf(stderr, "%d must be >=0\n", atoi(argv[1]);
     return -1;
  }
  pthread_attr_init(&attr); /* get the default attributes */
  pthread_create(&tid, &attr, runner, argv[1]); /* create the thread */
 /*the main thread can do other work parallelly with the new thread
here*/
  pthread_join(tid, NULL);     /* wait for the thread to exit */
  printf("sum= %d\n", sum);
}
```

```
/* The thread will begin
control in this function */
void *runner(void *param)
{
    int i, upper=atoi(param);
    sum = 0;
    for(i = 1; i <= upper; i++)
        sum+=i;
    pthread_exit(0);
}
```

观察线程的行为

```c
#include <stdio.h>
#include <pthread.h>
#include "mythreads.h"

static volatile int counter = 0;

//
// mythread()
//
// Simply adds 1 to counter repeatedly, in a loop
// No, this is not how you would add 10,000,000 to
// a counter, but it shows the problem nicely.
//
void *
mythread(void *arg)
{
    printf("%s: begin\n", (char *) arg);
    int i;
    for (i = 0; i < 1e7; i++) {
        counter = counter + 1;
    }
    printf("%s: done\n", (char *) arg);
    return NULL;
}

//
// main()
//
// Just launches two threads (pthread_create)
// and then waits for them (pthread_join)
//
int
main(int argc, char *argv[])
{
    pthread_t p1, p2;
    printf("main: begin (counter = %d)\n", counter);
    Pthread_create(&p1, NULL, mythread, "A");
    Pthread_create(&p2, NULL, mythread, "B");

    // join waits for the threads to finish
    Pthread_join(p1, NULL);
    Pthread_join(p2, NULL);
    printf("main: done with both (counter = %d)\n", counter);
    return 0;
}
```

线程编译时的注意点

- 需要链接线程库
  - gcc –lpthread –o testthread testthread.c

Posix 无名信号量

- 初始化：

  int sem_init(sem_t *sem, int pshared, unsigned value);

- 销毁：

  Int sem_destroy(sem_t *sem);

- PV操作

  P操作：int sem_wait(sem_t *sem);

  V操作：int sem_post(sem_t *sem);

Posix 无名信号量实现互斥

```
#include <semphore.h>
sem_t mutex;   //声明互斥信号量变量；
sem_init(&mutex, 0, 1);  //初始化其值为1
sem_wait(&mutex);    //P操作
sem_post(&mutex);   //V操作
```

实验任务2-多线程实现单词统计

- 有两个文件，现要求统计所有文件中的单词个数。为了加快统计速度，可以使用多线程机制，为每个要统计的文件创建一个线程，用于检测该文件中的单词个数。
- 区分单词的原则：凡是一个非字母或数字的字符跟在字母或数字的后面，那么这个字母或数字就是单词的结尾

两种解法

- 解法1：
  - 两个线程共享一个全局变量total_words
  - 当一个线程扫描到一个单词时，就更新total_words变量
  - 对total_words变量的更新必须互斥访问，因此，需要使用semaphore信号量
  - 最后主程序输出total_words即可
- 解法2：
  - 两个变量不共享变量，各自独自统计各自文件的单词个数
  - 当线程结束后，将统计的单词个数返回给主线程（注意，作为返回值的变量不能是定义在线程函数中的局部变量）
  - 主程序等两个线程都结束后，将两个返回值相加，得到单词总数

解法一：程序框架

```
#include <stdio.h>
#include <pthread.h>
#include <ctype.h>
#include <semaphore.h>
sem_t mutex;            //访问total_words的互斥信号量；
int total_words = 0;  //共享变量，线程应互斥访问
void *count_words(void *);
```

```
int main(int ac, char *av[]){   //av[1], av[2]分别存放两个待统计文件的文件名
   if(ac!=3){
       printf("Usage:%s file1 file2\n", av[0]);
       exit(1);
   }
   /*初始化信号量mutex的值为1*/
  /*分别以av[1], av[2]为参数，创建两个线程t1, t2 */
  /* 让主线程等待线程t1和t2完成后再执行*/
  /*输出统计出来的单词总数*/
  /*销毁信号量*/
}
```

```
void *count_words(void *arg)
{
   char *filename = (char*)arg;     //传给该线程的文件名;
   FILE *fp;
   int c, prevc='\0';
   if((fp=fopen(filename, "r"))!=NULL){
       while((c=getc(fp))!=EOF)
       {
           if(!isalnum(c)&&isalnum(prevc)){    //isalnum()函数用于测试是否字母数字
               /*对互斥变量mutex进行P操作*/
                total_words++;
               /*对互斥变量进行V操作*/
            }
           prevc = c;
       }
       fclose(fp);
   }
   else
       perror(filename);
   return NULL;
}
```

解法二：程序框架

```
#include <stdio.h>
#include <pthread.h>
#include <ctype.h>
void *count_words(void *);
struct buf{   //该结构体用于向线程传递文件名，并存放返回值；
   char *filename;
   int wc_count;
}args[2];
```

```
int main(int ac, char *av[]){
   if(ac!=3){
       printf("Usage:%s file1 file2\n", av[0]);
       exit(1);
   }
   args[0].filename = av[1];
   args[0].wc_count = 0;
   args[1].filename = av[2];
   args[1].wc_count = 0;
  /*分别以args[0], args[1]为参数，创建两个线程t1, t2 */
  /* 让主线程等待线程t1和t2完成后再执行*/
  /* 分别获取两个线程的返回值*/
  /*输出统计出来的单词总数*/
```

```
void *count_words(void *arg)
{
   struct buf *ptr = (struct buf *)arg;
   char *filename = ptr->filename;
   FILE *fp;
   int c, prevc='\0';
   if((fp=fopen(filename, "r"))!=NULL){
      while((c=getc(fp))!=EOF)
      {
         if(!isalnum(c)&&isalnum(prevc))
            ptr->wc_count++;

         prevc = c;
      }
      fclose(fp);
   }
   else
      perror(filename);
   pthread_exit((void*)&ptr->wc_count);   //通过pthread_exit()退出可以向等
                                          //待该线程结束的pthread_join提供返回值

}
```