实验二：线程建立和销毁

函数参考手册：

#include <pthread..h>

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

返回值：若成功返回 0，出错返回错误码

函数说明：

thread 是返回线程的指针

start_routine 是执行的线程函数

arg 是函数参数

#include <pthread.h>

```
int pthread_join(pthread_t    th，void    **thread_return);
```

调用 pthread_join 的线程将挂起，一直到 th 线程结束

thread_return 如果不是 NULL，则 th 线程的返回值将被存储在 thread_return 中。

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

调用他的线程终止执行。retval 是线程的返回值。

综上所述，线程的创建是用下面的几个函数来实现的。

int pthread_create(pthread_t *thread,pthread_attr_t *attr,

```
void *(*start_routine)(void *),void *arg);
void pthread_exit(void *retval);
int pthread_join(pthread *thread,void **thread_return);
```

pthread_create 创建一个线程，thread 是用来表明创建线程的 ID，attr 指出线程创建时候的属性，我们用 NULL 来表明使用缺省属性。start_routine 函数指针是线程创建成功后开始执行的函数，arg 是这个函数的唯一一个参数，表明传递给 start_routine 的参数。pthread_exit 函数和 exit 函数类似用来退出线程，这个函数结束线程,释放函数的资源，并在最后阻塞，直到其他线程使用 pthread_join 函数等待它。然后将*retval 的值传递给**thread_return。由于这个函数释放所以的函数资源，所以 retval 不能够指向函数的局部变量。pthread_join 和 wait 调用一样用来等待指定的线程。

实验样例：

例 1：创建一个不带参数的线程，假设该程序文件名叫 thread1.c。

```
#include <pthread.h>
int helloworld1()   //线程体代码
{
   printf("hello world\n");
}

int main()
{
    pthread_t     t;
   //创建一个新的线程，参数传递 NULL
    pthread_create(&t, NULL, (void*)helloworld1, NULL);
    pthread_join(t, NULL);   //等待线程执行完毕
    exit(0);
}
```

编译该程序：

```
[liuhui@RedHat ex_thread]$   cc   thread1.c   –o   thread1   -lpthread
```

运行该程序：

```
[liuhui@RedHat ex_thread]$   ./thread1
```

例 2：创建一个带参数的线程，假设该程序文件名叫 thread2.c

```
#include <pthread.h>
int helloworld2(char* name)
```

```
{
   printf("hello world, %s\n", name);
}
int main()
{
     pthread_t      t;
     char s[]="liuhui" ;
     pthread_create(&t, NULL, (void*)helloworld2, s);
     pthread_join(t, NULL);
     exit (0);
}
```

编译该程序：

```
[liuhui@RedHat ex_thread]$    cc    thread2.c    –o    thread2    -lpthread
```

运行该程序：

```
[liuhui@RedHat ex_thread]$    ./thread2
```

上机练习：

Ex－1：运行例 1，观察结果，分析结果的成因。

Ex－2：运行例 2,，观察结果，分析结果的成因。

Ex－3：看下面的例子，找出错误的地方。

```
#include <pthread.h>
int helloworld3(int    * no)
{
   printf("hello world, %d\n", *no);
}
int main()
{
     pthread_t      t;
     int    no = 9;
     pthread_create(&t, NULL, (void*)helloworld3, no);
     pthread_join(t, NULL);
     exit (0);
}
```

Ex－4：看下面的例子，找出错误的地方。

```
#include <pthread.h>
int helloworld3(int      no)
{
   printf("hello world, %d\n",no);
}
```

```
int main()
{
    pthread_t      t;
    int    no = 9;
    pthread_create(&t, NULL, (void*)helloworld3, &no);
    pthread_join(t, NULL);
    exit (0);
}
```

Ex－5：自己编写一个程序，验证子线程和创建它的进程共享进程的状态和资源。要求创建带一个整型参数的子线程，它在屏幕上输出这个整型数值；主线程则用一个循环不断累加传给子线程的这个整数，编写程序观察结果。