1. 线程相关概念

- 自行安装线程man page,命令: sudo apt-get install manpages-posix-dev
- 察看指定线程的LWP号:
    - 线程号和线程ID是有区别的
    - 线程号是给内核看的
    - 查看方式:
        - 找到程序的进程ID
        - ps -Lf pid

        

        明显的多线程的实现,这里可以看出,这个pid是一样的,因为共用一块内存的地址空间

2. 进程和线程的区别:

3. 线程操作函数
    1. 创建线程 -- pthread_create
    - 函数原型:

// 如果成功0,失败返回错误号 // perror() 不能使用该函数打印错误信息

```
if(ret != 0)
    {
        printf("error number%d\n",ret);
        printf("%s\n",strerror(ret));
    }
```

注意打印错误信息的方式

int pthread_create(pthread_t *thread, // 线程ID=无符号长 整形 const pthread_attr_t *attr, // 线程属性,NULL 设置是否分离 void *(*start_routine) (void *), // 线程处理函数 void *arg // 线程处理函数参数);

- 参数:
    - thread: 传出参数, 线程创建成功之后,会被设置一个合适的值会被设置一个合适的值
    - attr: 默认传NULL
    - start_routine: 子线程的处理函数
    - arg: 回调函数的参数
- 主线程先退出, 子线程会被强制结束
- 验证线程之间共享全局变量

```
//
// Created by bruce on 18-5-20.

//
```

```c
#include <sys/types.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

//
// Created by bruce on 18-5-20.
//
void *myfunc(void* args)
{
    int num = (int)args;
    printf("%dth child thread id:%lu\n" ,num,pthread_self());
    return NULL;
}
int main()
{
    //穿件一个子线程
    //线程ID变量
    pthread_t pthid[5];
    for(int i=0;i<5;++i)
    {
        //第四个参数传递的是地址
//        pthread_create(&pthid[i],NULL,myfunc,(void*)&i);
        //强制性转为int,然后传值进去.
        pthread_create(&pthid[i],NULL,myfunc,(void*)i);
    }
    printf("parent thread id:%lu\n" ,pthread_self());

    for(int i=0;i<5;++i)
    {
        printf("i=%d\n",i);
    }
    sleep(2);
    return 0;
}
```

2. 单个线程退出

   单个线程退出 -- pthread_exit

   - exit(0);
   - 函数原型: void pthread_exit(void *retval);
     - retval指针:必须指向全局,堆
   - 如果是主线程退出了,那么对子线程没有影响.
   - 子线程的退出不能用exit会对主线程有影响.
   - 最好使用pthread_exit来退出.

```c
//
// Created by bruce on 18-5-20.
//
```

```c
#include <sys/types.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

//
// Created by bruce on 18-5-20.
//
void *myfunc(void* args)
{
    printf("child thread id:%lu\n" ,pthread_self());
    for(int i =0;i<5;i++)
    {
        printf("child i=%d\n",i);
        if(i==2)
        {
            pthread_exit(NULL);
        }
    }
    return NULL;
}
int main()
{
    //穿件一个子线程
    //线程ID变量
    pthread_t pthid;

    int ret = pthread_create(&pthid,NULL,myfunc,NULL);

    if(ret != 0)
    {
        printf("error number%d\n",ret);
        printf("%s\n",strerror(ret));
    }
    printf("parent thread id:%lu\n" ,pthread_self());

    int i=0;
    while(1)
    {
        printf("i=%d\n",i++);
    }
    pthread_exit(NULL);
    sleep(2);
    return 0;
}
```

3. 主线程阻塞等待线程退出, 获取线程退出状态 --pthread_join

- 函数原型:
- int pthread_join(pthread_t thread, void**retval);
  - thread:要回收的子线程的线程id
  - retval:读取线程退出的时候携带的状态信息
    - 传出参数
    - void* ptr;
    - pthread_join(pthid, &ptr);
    - 指向的内存和pthread_exit参数指向同一块内存地址

```
#include <sys/types.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

int num =100;
void *myfunc(void* args)
{
    printf("child thread id:%lu\n",pthread_self());
    for(int i =0;i<5;i++)
    {
        printf("child i=%d\n",i);
        if(i==2)
        {
            pthread_exit(&num);
        }
    }
    return NULL;
}
int main()
{
    //穿件一个子线程
    //线程ID变量
    pthread_t pthid;

    int ret = pthread_create(&pthid,NULL,myfunc,NULL);

    if(ret != 0)
    {
        printf("error number%d\n",ret);
        printf("%s\n",strerror(ret));
    }
    printf("parent thread id:%lu\n",pthread_self());
    void *ptr = NULL;
    pthread_join(pthid,&ptr); //阻塞函数
    printf("number = %d\n",*(int*)ptr);
    int i=0;
    while(i<10)
    {
```

```
        printf("i=%d\n",i++);
    }
    pthread_exit(NULL);
    sleep(2);
    return 0;
}
```

4. 线程分离 -- pthread_detach
    - 函数原型: int pthread_detach(pthread_t thread);
    - 调用该函数之后不需要pthread_join
    - 子线程会自动回收自己的pcb
5. 杀死(取消)线程 -- pthread_cancel
    - 函数原型: int pthread_cancel(pthread_t thread);
    - 使用注意事项:
    - 在要杀死的子线程对应的处理的函数的内部,必须做过一次系统调用(即是所谓的取消点)
        - write read printf
        - pthread_testcancel(); // 来设置一个取消点
        - 没有取消点:
            - int a;
            - a = 2;
            - int b = a+3;
6. 比较两个线程ID是否相等(预留函数) --pthread_equal
    - 函数原型:

    int pthread_equal(pthread_t t1, pthread_t t2);

    没有必要使用

线程属性

通过属性设置线程的分离

1. 线程属性类型: pthread_attr_t attr;
2. 线程属性操作函数:
   ○ 对线程属性变量的初始化
   ▪ int pthread_attr_init (pthread_attr_t* attr);
   ○ 设置线程分离属性
   ▪ int pthread_attr_setdetachstate (
   pthread_attr_t* attr,
   int detachstate
   );
   ▪ 参数:
   □ attr: 线程属性
   □ detachstate:
   ⛀ PTHREAD_CREATE_DETACHED(分离)
   ⛀ PTHREAD_CREATE_JOINABLE(非分离)

- 释放线程资源函数
- int pthread_attr_destroy(pthread_attr_t *attr);

```
#include <sys/types.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>

//
// Created by bruce on 18-5-20.
//
void *myfunc(void* args)
{
    printf("child thread id:%lu\n",pthread_self());
    return NULL;
}
int main()
{
    //穿件一个子线程
    //线程ID变量
    pthread_t pthid;
    //初始化线程的属性
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    //设置分离
    pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED);
    int ret = pthread_create(&pthid,&attr,myfunc,NULL);

    if(ret != 0)
    {
        printf("error number%d\n",ret);
        printf("%s\n",strerror(ret));
    }
    printf("parent thread id:%lu\n",pthread_self());

    for(int i=0;i<5;++i)
    {
        printf("i=%d\n",i);
    }
    sleep(2);
    pthread_attr_destroy(&attr);
    return 0;
}
```

线程数据叔叔:

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <pthread.h>

#define MAX 10000
// 全局变量
int number;

// 线程处理函数
void* funcA_num(void* arg)
{
    for(int i=0; i<MAX; ++i)
    {
        int cur = number;
        cur++;
        number = cur;
        printf("Thread A, id = %lu, number = %d\n", pthread_self(), number);
        usleep(10);
    }

    return NULL;
}

void* funcB_num(void* arg)
{
    for(int i=0; i<MAX; ++i)
    {
        int cur = number;
        cur++;
        number = cur;
        printf("Thread B, id = %lu, number = %d\n", pthread_self(), number);
        usleep(10);
    }

    return NULL;
}

int main(int argc, const char* argv[])
{
    pthread_t p1, p2;

    // 创建两个子线程
    pthread_create(&p1, NULL, funcA_num, NULL);
    pthread_create(&p2, NULL, funcB_num, NULL);

    // 阻塞，资源回收
    pthread_join(p1, NULL);
    pthread_join(p2, NULL);
```

```
    return 0;
}
```