1、创建一个子线程，传入数值1，在子线程中能够获取并打印，子线程退出，返回数值2，主线程通过pthread\_join获取等待子线程结束并获取子线程的退出值并打印



 #include <func.h>

void \* thread\_func(void \*p)

{

    printf("i am child thread, get %d \n",\*(int \*)p);

    pthread\_exit((void \*)2);

}

int main()

{

    int ret ,i=1;

    //创建子线程

    pthread\_t pthid;

    ret = pthread\_create(&pthid,NULL,thread\_func,&i);

    THREAD\_ERR\_CHECK(ret,"pthread\_create");

    //等待子线程

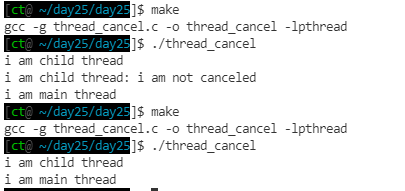
    pthread\_join(pthid,(void \*\*)&ret);

    printf("i am main thread, child return %d \n", ret);

    return 0;

}

2、创建一个子线程，主线程cancel子线程



#include <func.h>

void \* thread\_func(void \*p)

{

    printf("i am child thread\n");

    sleep(1);

    printf("i am child thread: i am not canceled\n");

}

int main()

{

    int ret;

    //创建子线程

    pthread\_t pthid;

    ret = pthread\_create(&pthid,NULL,thread\_func,NULL);

    THREAD\_ERR\_CHECK(ret,"pthread\_create");

    ret = pthread\_cancel(pthid);

    THREAD\_ERR\_CHECK(ret,"pthread\_cancel");

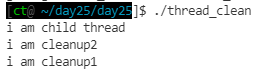
    pthread\_join(pthid,NULL);

    printf("i am main thread\n");

    return 0;

}

3、创建一个子线程，cancel子线程，在子线程中push两个清理函数，感受清理函数的执行顺序



#include <func.h>

void cleanup(void \*p)

{

    printf("i am cleanup%ld\n",(long)p);

}

void \* thread\_func(void \*p)

{

    pthread\_cleanup\_push(cleanup,(void \*)1);

    pthread\_cleanup\_push(cleanup,(void \*)2);

    //cancel点，异步

    printf("i am child thread\n");

    sleep(1);

    printf("i am child thread: i am not canceled\n");

    pthread\_cleanup\_pop(1);  //0：pop了但什么都不做  1：pop并执行清理函数

    pthread\_cleanup\_pop(1);

}

int main()

{

    int ret;

    //创建子线程

    pthread\_t pthid;

    ret = pthread\_create(&pthid,NULL,thread\_func,NULL);

    THREAD\_ERR\_CHECK(ret,"pthread\_create");

    ret = pthread\_cancel(pthid);

    THREAD\_ERR\_CHECK(ret,"pthread\_cancel");

    pthread\_join(pthid,NULL);

    return 0;

}

4、子线程，主线程，同时对一个变量加2千万，通过加锁，实现最终效果是4千万。



#include <func.h>

#define N 20000000

typedef struct

{

    int num;

    pthread\_mutex\_t mutex;

} data\_t;

void \* thread\_func(void \*p)

{

    data\_t \*d = (data\_t \*)p;

    for(int i=0;i<N;i++)

    {

        pthread\_mutex\_lock(&d->mutex);

        d->num++;

        pthread\_mutex\_unlock(&d->mutex);

    }

    pthread\_exit(NULL);

}

int main()

{

    int ret;

    //初始化锁mutex

    pthread\_mutex\_t mutex;

    ret = pthread\_mutex\_init(&mutex,NULL);

    THREAD\_ERR\_CHECK(ret,"pthread\_mutex\_init");

    //初始化结构体数据d

    data\_t d;

    d.num = 0;

    d.mutex = mutex;

    //创建进程

    pthread\_t pthid;

    ret = pthread\_create(&pthid,NULL,thread\_func,&d);

    THREAD\_ERR\_CHECK(ret,"pthread\_create");

    //main 线程加2千万

    for(int i=0;i<N;i++)

    {

        pthread\_mutex\_lock(&d.mutex);

        d.num++;

        pthread\_mutex\_unlock(&d.mutex);

    }

    //等待子线程

    pthread\_join(pthid,NULL);

    printf("d.num = %d \n", d.num);

    //销毁锁mutex

    ret = pthread\_mutex\_destroy(&d.mutex);

    THREAD\_ERR\_CHECK(ret,"pthread\_mutex\_destroy");

    return 0;

}

5、通过设置线程锁属性，用mutex实现两个进程各加2千万，最终实现4千万。



#include <func.h>

#define N 20000000

typedef struct

{

    int num;

    pthread\_mutex\_t mutex;

}data\_t;

int main()

{

    //共享内存与连接

    int shmid;

    shmid = shmget(1000,4096,IPC\_CREAT|0600);

    ERROR\_CHECK(shmid,-1,"shmid");

    data\_t \*p = shmat(shmid,NULL,0);

    ERROR\_CHECK(shmid,(data\_t \*)-1,"shmid");

    //定义并设置锁属性

    pthread\_mutexattr\_t mutexattr;

    pthread\_mutexattr\_init(&mutexattr);

    pthread\_mutexattr\_setpshared(&mutexattr,PTHREAD\_PROCESS\_SHARED);

    //初始化结构指针p

    p->num = 0;

    pthread\_mutex\_init(&p->mutex,&mutexattr);

    int i;

    if (!fork())

    {

        for (i=0;i<N;i++)

        {

            pthread\_mutex\_lock(&p->mutex);

            p->num++;

            pthread\_mutex\_unlock(&p->mutex);

        }

        return 0;

    }

    else

    {

        for (i=0;i<N;i++)

        {

            pthread\_mutex\_lock(&p->mutex);

            p->num++;

            pthread\_mutex\_unlock(&p->mutex);

        }

        wait(NULL);

        printf("p->num=%d\n", p->num);

        shmctl(shmid,IPC\_RMID,NULL);

        return 0;

    }

}

6、编写火车站买票，两个子线程卖票，确保没有卖负票

#include <func.h>

typedef struct

{

    int num;

    pthread\_mutex\_t mutex;

}data\_t;

void \* sale\_tickets1(void \*p)

{

    data\_t \*d = (data\_t \*)p;

    while(1)

    {

        pthread\_mutex\_lock(&d->mutex);

        if (d->num > 0)

        {

            printf("i am sale\_tickets1, start sale\n");

            d->num--;

            printf("i am sale\_tickets1, end sale, remain %d\n", d->num);

            pthread\_mutex\_unlock(&d->mutex);

        } else {

            pthread\_mutex\_unlock(&d->mutex);

            break;

        }

    }

}

void \* sale\_tickets2(void \*p)

{

    data\_t \*d = (data\_t \*)p;

    while(1)

    {

        pthread\_mutex\_lock(&d->mutex);

        if (d->num > 0)

        {

            printf("i am sale\_tickets2, start sale\n");

            d->num--;

            printf("i am sale\_tickets2, end sale, remain %d\n", d->num);

            pthread\_mutex\_unlock(&d->mutex);

        } else {

            pthread\_mutex\_unlock(&d->mutex);

            break;

        }

    }

}

int main()

{

    int ret;

    data\_t d;

    d.num = 20;

    pthread\_mutex\_init(&d.mutex,NULL);

    //创建子线程

    pthread\_t pthid[2];

    ret = pthread\_create(&pthid[0],NULL,sale\_tickets1,(void \*)&d);

    THREAD\_ERR\_CHECK(ret,"pthread\_create1");

    ret = pthread\_create(&pthid[1],NULL,sale\_tickets2,(void \*)&d);

    THREAD\_ERR\_CHECK(ret,"pthread\_create2");

    for(int i=0;i<2;i++)

    {

        pthread\_join(pthid[i],NULL);

    }

    printf("i am main thread, remain %d tickets\n", d.num);

    return 0;

}