# 线程

2024年6月1日　　21:26



## 五、线程

与进程类似，线程(thread)是允许程序并发执行多任务的一种机制。**一个进程可以包含多个线程**，每个线程会独立执行相同的程序代码，且共享同一份全局内存区域。

同一进程中的多个线程可以并发执行。在多处理器环境下，多个线程可以同时并行。如果一个线程因等待 1/0 操作而遭阻塞，那么其他线程依然可以继续运行。

**线程相对进程的优势:**

进程间的信息难以共享，需要进程间通信。线程间能方便、快速的共享信息，不过要避免多个线程同时修改同一数据的情况。

调用fork()创建进程的代价较高，需要复制父进程的内容。线程比进程的创建要快很多。

### 5.1 创建线程

启动程序时，产生的进程只有单条线程，称之为初始(initial)或主(main)线程。函数pthread create()负责创建一条新线程。新线程通过调用带有参数 arg 的函数 start routine(即 start routine(arg))而开始执行。调用 pthread create()的线程会继续执行该调用之后的语句。

进程内部的每个线程都有一个唯一标识，称为线程 ID。线程获取自己的线程 ID 使用pthread self()函数。

使用命令 ps -eLf 查看线程

ubuntu20 编译有线程的代码可能失败，使用 gcc 编译，加参数 -l pthread

**pthread_create函数**

**函数描述:**

创建一个新线程。类似进程中的fork() 函数,

**头文件:**

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

**函数原型:**

```
int pthread create(pthread t*thread,
                   const pthread attr_t *attr,
                   void*(*start_routine)(void *),
                   void *arg);
```

**函数参数:**

thread:传出参数，保存系统为我们分配好的线程ID

attr:通常传 NULL，表示使用线程默认属性。若想使用具体属性也可以修改该参数。

start_routine:函数指针，指向线程主函数(线程体)，该函数运行结束，则线程结束

线程主函数的参数:arg

**函数返回值:**

成功返回 0
失败返回错误号

**pthread_self函数**

**函数描述:**

获取线程 ID。类似于进程中的getpid()函数。线程ID 是进程内部的识别标志(两个进程间，允许线程ID相同)

**函数原型:**

pthread t pthread_self(void),

**函数返回值:**

成功返回线程ID，pthreadt为无符号整型(%lu)
失败无

## 5.2 线程终止

**终止线程的方式:**

(1)线程 start_routine 函数执行 return 。
(2)线程调用 pthread_exit()终止线程。
(3)调用 pthread_cancel(thread)取消指定线程
(4)任意线程调用了 exit()，或者主线程执行了retur 语句(在 main()函数中)，都会导致进程中的所有线程立即终止

pthread_exit()函数将终止调用线程，且其返回值可由另一线程通过调用 pthread_join()来获取。调用 pthread exit()相当于在线程的 start routine 函数中执行return，不同之处在于可在线程 start routine 函数所调用的任意函数中调用 pthread exit()都能够直接终止线程。

**pthread_exit函数**

**函数描述:**

终止线程

**函数原型:**

void pthread exit(void *retval);

**函数参数:**

retval:表示线程退出状态，通常传 NULL

**调用pthread_exit()线程结束:**

```
int a = 0;
void* thread1(void* arg)
{
    printf("thread1 id = %lu\n",pthread_self());
    while(a < 10)
    {
        printf("a = %d\n",a++);
        sleep(1);
        pthread_exit(NULL);
    }
    return NULL;
}
int main(int argc, char* argv[])
{
    pthread_t mpd;

    pthread_create(&mpd,NULL,thread1,NULL);
    printf("main id = %lu\n",pthread_self());
    sleep(10);

    return 0;

}
```

**主函数return，线程终止**

```
// 主函数return，线程终止
int a = 0;
void* thread1(void* arg)
```

```
{
    printf("thread1 id = %lu\n",pthread_self());
    while(a < 10)
    {
        printf("a = %d\n",a++);
        sleep(5);
    }
    return NULL;
}
int main(int argc, char* argv[])
{
    pthread_t mpd;

    pthread_create(&mpd,NULL,thread1,NULL);
    printf("main id = %lu\n",pthread_self());

    return 0;
    // exit(1);
}
```

## pthread_cancel函数

**函数描述:**

向指定线程发送一个取消请求。发出取消请求后，函数pthread cancel()当即返回不会等待目标线程的退出。被请求取消的线程不会立即取消，需要等待到达某个取消点取消点通常是一些系统调用，也可以使pthread_testcancel 函数手动创建一个取消点，被取消的线程返回值是 PTHREAD CANCELED(-1)。

**函数原型:**

int pthread cancel(pthread t thread).

**函数参数:**

thread:要取消的线程ID

**函数返回值:**

成功返回 0

失败返回错误号

```
// pthread_cancel函数
int a = 0;
void* thread1(void* arg)
{
    while(1)
    {
        // a 在0~10循环
        if(a==10)
        {
            a = 0;
        }
        a++;

        //printf("hello\n"); // 取消点
        pthread_testcancel();//取消点
    }
    return NULL;
}
int main(int argc, char* argv[])
{
    pthread_t mpd;

    pthread_create(&mpd,NULL,thread1,NULL);
    // 取消线程，不会立即取消，得等待线程1中执行取消点（系统调用）
    int ret = pthread_cancel(mpd);
    if(ret != 0)
    {
        printf("pthread_cancel error, ret = %d\n",ret);
    }
    else
    {
        printf("pthread_cancel success, ret = %d\n",ret);
    }
    while(1)
    {
        printf("ret = %d\n",ret);
        sleep(1);
    }
```

```
    return 0;
}
```

## 5.3 线程的链接与分离

### pthread_join 函数

**函数描述:**

等待指定线程终止并回收，这种操作叫做连接(joining)。未连接的线程会产生僵尸线程，类似僵尸进程的概念。

**函数原型:**

```
int pthread join(pthread t thread, void **retval);
```

**函数参数:**

thread:要等待的线程 ID

retval:存储线程返回值

**函数返回值:**

成功返回 0

失败返回错误号

### pthread detach函数

**函数描述:**

默认线程是可连接的(ioinable)当线程退出时，其他线程可以通过调用pthread join()获取其返回状态。有时，程序员并不关心线程的返回状态，只是希望系统在线程终止时能够自动清理并移除。这时可以调用 pthread_detach()函数，将线程标记为分离(detached)状态。

**函数原型:**

```
int pthread detach(pthread t thread);
```

**函数参数:**

thread:要分离的线程ID

retval:存储线程返回值

**函数返回值:**

成功返回 0

失败返回错误号

### 练习

定义一个全局变量 a，创建 10 个线程，每个线程对a进行自增(a++)100万次，所有线程自增结束主线程打印 a的值。

```
int a = 0;
void* thread1(void* arg)
{
    printf("thread1 id = %lu\n",pthread_self());
    for(int i = 0; i < 1000000; i++)
    {
        a++;
    }
    return NULL;
}
int main(int argc, char* argv[])
{
    pthread_t mpd;
    pthread_t mpd_Arr[10];
    for(int i = 0; i < 10; i++)
    {
        pthread_create(&mpd_Arr[i],NULL,thread1,NULL);
    }
    for(int i = 0; i < 10; i++)
    {
        pthread_join(mpd_Arr[i],NULL);
    }
```

```
    printf("main id = %lu\n",pthread_self());
    // sleep(10);
    printf("a = %d\n",a); // a!=100w
    return 0;
}
```