操作系统

时间：2023.10.4

1. 小林coding：如何避免死锁？

1.1 死锁的概念

多个进程/线程都在等待对方释放锁，在没有外力介入的情况下，这些进程/线程会一直相互等待，没办法执行下去，这种情况就是死锁。

1.2 死锁发生的条件

1. 互斥条件

- 多个进程/线程不能同时使用同一个资源。

2. 持有并等待条件

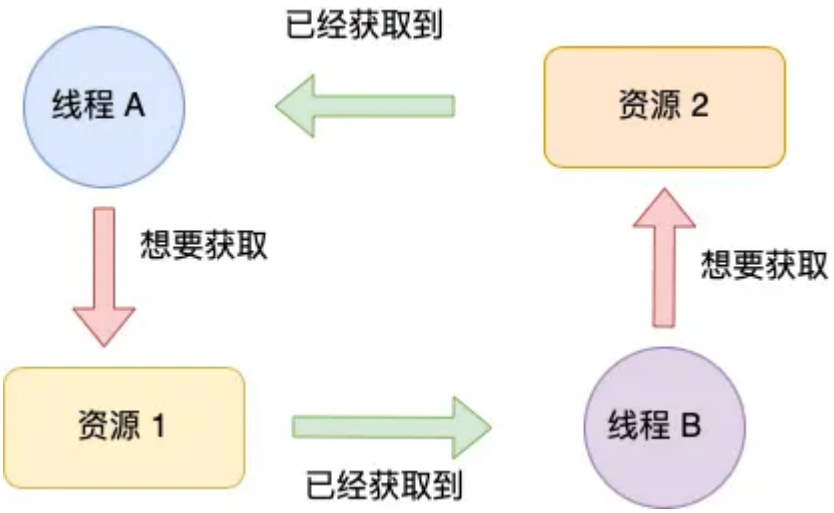
- 某个进程/线程阻塞等待某个资源的同时，还持有另外一个资源。

3. 不可剥夺条件

- 线程持有的资源在自己用完之前是不能被其他线程获取的。

4. 环路等待条件

- 多个线程获取资源的顺序构成了环形链。



1.3 模拟死锁问题的产生

- 创建 2 个线程，分别为线程 A 和 线程 B
- 然后有两个互斥锁，分别是 mutex_A 和 mutex_B

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_B = PTHREAD_MUTEX_INITIALIZER;

int main()
{
    pthread_t tidA, tidB;

    //创建两个线程
    pthread_create(&tidA, NULL, threadA_proc, NULL);
    pthread_create(&tidB, NULL, threadB_proc, NULL);

    pthread_join(tidA, NULL);
    pthread_join(tidB, NULL);

    printf("exit\n");

    return 0;
}
```

```
//线程 A 函数
void *threadA_proc(void *data)
{
    printf("thread A waiting get ResourceA \n");
    pthread_mutex_lock(&mutex_A);
    printf("thread A got ResourceA \n");

    sleep(1);

    printf("thread A waiting get ResourceB \n");
    pthread_mutex_lock(&mutex_B);
    printf("thread A got ResourceB \n");

    pthread_mutex_unlock(&mutex_B);
    pthread_mutex_unlock(&mutex_A);
    return (void *)0;
}
```

- 线程 A 函数的过程：
  1. 先获取互斥锁 A，然后睡眠 1 秒；
  2. 再获取互斥锁 B，然后释放互斥锁 B；
  3. 最后释放互斥锁 A；

```
//线程B函数
void *threadB_proc(void *data)
{
    printf("thread B waiting get ResourceB \n");
    pthread_mutex_lock(&mutex_B);
    printf("thread B got ResourceB \n");

    sleep(1);

    printf("thread B waiting  get ResourceA \n");
    pthread_mutex_lock(&mutex_A);
    printf("thread B got ResourceA \n");

    pthread_mutex_unlock(&mutex_A);
    pthread_mutex_unlock(&mutex_B);
    return (void *)0;
}
```

- 线程 B 函数的过程：
  1. 先获取互斥锁 B，然后睡眠 1 秒；
  2. 再获取互斥锁 A，然后释放互斥锁 A；
  3. 最后释放互斥锁 B；

运行这个程序，运行结果如下:

thread B waiting get ResourceB thread B got ResourceB thread A waiting get ResourceA thread A got ResourceA thread B waiting get ResourceA thread A waiting get ResourceB // 阻塞中...

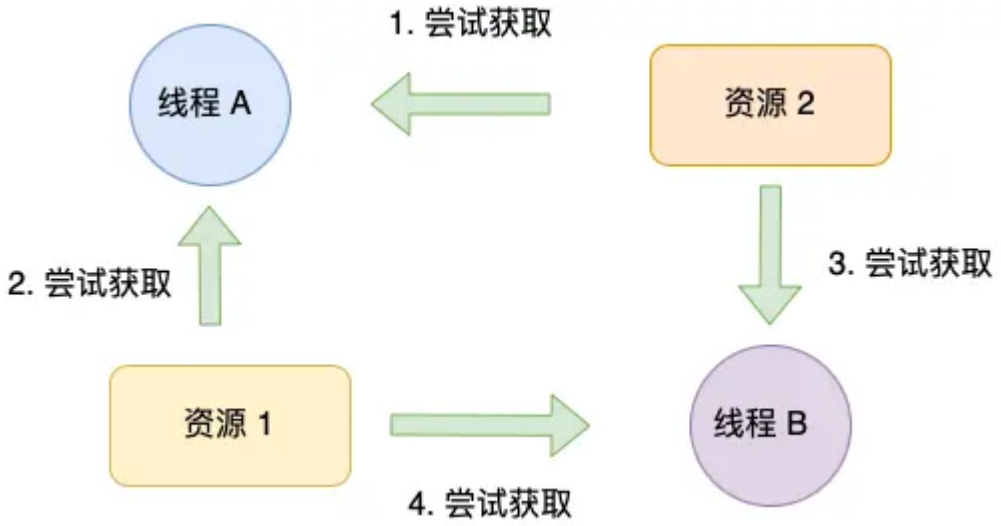
线程 B 在等待互斥锁 A 的释放，线程 A 在等待互斥锁 B 的释放，双方都在等待对方资源 的释放，很明显，产生了死锁问题。

1.4 避免死锁问题的产生

办法：破坏产生死锁的条件中的任何一个条件即可，最常见的并且可行的就是**使用资源有序分配法，来破坏环路等待条件。**

1.4.1 什么是资源有序分配法?

线程 A 和 线程 B 总是以相同的顺序申请自己想要的资源。



1.4.2 修改1.3代码消除死锁

- 修改后：

```
//线程 B 函数访问资源的顺序，同线程 A 一样，先获取互斥锁 A，然后获取互斥锁 B
void *threadB_proc(void *data)
{
    printf("thread B waiting get ResourceA \n");
    pthread_mutex_lock(&mutex_A);
    printf("thread B got ResourceA \n");

    sleep(1);

    printf("thread B waiting  get ResourceB \n");
    pthread_mutex_lock(&mutex_B);
    printf("thread B got ResourceB \n");

    pthread_mutex_unlock(&mutex_B);
    pthread_mutex_unlock(&mutex_A);
    return (void *)0;
}
```

- 线程 B 函数的过程：
  1. 先获取互斥锁 A，然后睡眠 1 秒；
  2. 再获取互斥锁 B，然后释放互斥锁 B；
  3. 最后释放互斥锁 A；

修改前：

```
//线程B函数
void *threadB_proc(void *data)
{
 printf("thread B waiting get ResourceB \n");
 pthread_mutex_lock(&mutex_B);
 printf("thread B got ResourceB \n");

sleep(1);

printf("thread B waiting get ResourceA \n");
 pthread_mutex_lock(&mutex_A);
 printf("thread B got ResourceA \n");

pthread_mutex_unlock(&mutex_A);
 pthread_mutex_unlock(&mutex_B);
 return (void *)0;
}
```

- 线程 B 函数的过程：

  1. 先获取互斥锁 B，然后睡眠 1 秒；
  2. 再获取互斥锁 A，然后释放互斥锁 A；
  3. 最后释放互斥锁 B；