多线程以及多进程改善了系统资源的利用率并提高了系统的处理能力。然而，并发执行也带来了新的问题—— 死锁。

死锁是指多个线程因竞争资源而造成的一种僵局（互相等待），若无外力作用，这些进程都将无法向前推进。

**线程死锁产生的必要条件:**

1）**互斥条件**：一个资源每次只能被一个进程使用

2）**请求与保持条件**：一个进程因请求資源而阻塞时对已获得的资源保持不放。

3）**不可剥夺条件**：进程已获得的资源，在未使用完之前，不能强行剥夺。

4）**循环等待条件**：若干进程之间形成一种头尾相接的循环等待资源关系。

**银行家算法:**

避免死锁算法中最有代表性的算法是 Dijkstra EW于1968年提出的银行家算法，防止死锁的机构只能确保上述四个条件之一不出现，则系统就不会发生死锁。

银行家算法是一种最有代表性的避免死锁的算法。通过这个算法可以用来解决生活中的实际问题，如银行贷款等。银行家在客户申请的贷款应先计算此借款的安全性若借款不会导致银行进入不安全状态，则借出，否则不予借出。银行家就好比操作系统，资金就是资源，客户就相当于要申请資源的进程。

#include<process.h>

#include<windows.h>

#include<stdio.h>

CRITICAL\_SECTION cs1;

CRITICAL\_SECTION cs2;

void ThreadFun1(void \*param);

void ThreadFun2(void \*param);

//演示死锁发生

int main()

{

InitializeCriticalSection(&cs1);

InitializeCriticalSection(&cs2);

uintptr\_t t1= \_beginthread(ThreadFun1, 0, "A");

uintptr\_t t2 = \_beginthread(ThreadFun2, 0, "B");

//A在1区域，等着B从2区域出来

//B在2区域，等着A从1区域出来

//相互等待中，发生死锁

HANDLE hArr[] = { ( HANDLE)t1, (HANDLE)t2 };

WaitForMultipleObjects(2, hArr, true, INFINITE);

return 0;

}

void ThreadFun1(void \*param)

{

char \*name = (char \*)param;

//进入1区域之后，任何人无法进1区域

EnterCriticalSection(&cs1);

printf("%s进入了1区域,休息3秒\n", name);

Sleep(3000);

printf("%s想进入2区域\n", name);

EnterCriticalSection(&cs2);

LeaveCriticalSection(&cs2);

LeaveCriticalSection(&cs1);

}

void ThreadFun2(void \*param)

{

char \*name = (char \*)param;

//进入2区域之后，任何人无法进2区域

EnterCriticalSection(&cs2);

printf("%s进入了2区域,休息3秒\n", name);

Sleep(3000);

printf("%s想进入1区域\n", name);

EnterCriticalSection(&cs1);

LeaveCriticalSection(&cs1);

LeaveCriticalSection(&cs2);

}