临界区（Critical section）一次只允许一个线程访问资源，而信号量（Semaphore）允许N个线程在同一时刻访问同一资源。

* 使用 CreateSemaphore（）创建信号量时要指出允许的最大资源计数和当前可用资源计数。
* 一般将当前可用资源计数设置为最大值，每增加一个线程对资源的访问，可用资源计数减1，只要当前可用资源计数是大于0，就可发出信号。当可用计数减小到0则说明当前占用资源线程数已达所允许的最大值，不允许其他线程的进入，此时无法发出信号。
* 线程在处理完共享资源后，通过 ReleaseSemaphore（）  
  函数将当前可用资源计数加1。

**1. CreateSemaphore()**

[CreateSemaphoreA function (winbase.h) - Win32 apps | Microsoft Docs](https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createsemaphorea)

HANDLE CreateSemaphoreA(

LPSECURITY\_ATTRIBUTES lpSemaphoreAttributes,

LONG lInitialCount,

LONG lMaximumCount,

LPCSTR lpName

);

**2. ReleaseSemaphore()**

[ReleaseSemaphore function (synchapi.h) - Win32 apps | Microsoft Docs](https://docs.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-releasesemaphore)

BOOL ReleaseSemaphore(

HANDLE hSemaphore,

LONG lReleaseCount,

LPLONG lpPreviousCount

);

**3. WaitForSingleobject()**

[WaitForSingleObject function (synchapi.h) - Win32 apps | Microsoft Docs](https://docs.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-waitforsingleobject)

DWORD WaitForSingleObject(

HANDLE hHandle,

DWORD dwMilliseconds

);

该函数需要传递一个内核对象句柄，如果该内核对象处于未通知状态，则该函数导致线程进入阻塞状态；如果该内核对象处于已通知状态，则该函数立即返回 WAIT\_OBJECT()第二个参数指明要等待的时间（毫秒），INFINITE表示无限等待，如果第二个参数为0，那么函数立即返回。如果等待超时，该函数返 WAIT\_TIMEOUT如果该函数失败，返回 WAIT\_FAILED。

**4. CloseHandle()**

[CloseHandle function (handleapi.h) - Win32 apps | Microsoft Docs](https://docs.microsoft.com/en-us/windows/win32/api/handleapi/nf-handleapi-closehandle)

BOOL CloseHandle(

HANDLE hObject

);

**5. 示例代码**

#include<windows.h>

#include<stdio.h>

DWORD WINAPI ThreadFun(LPVOID paramter);

//车辆名称和停车时间

struct Car

{

char name[20];

DWORD time;

};

HANDLE hSemaphore = INVALID\_HANDLE\_VALUE;

int main()

{

//只有3个停车位资源

hSemaphore = CreateSemaphore(NULL, 3, 3, L"停车位");

HANDLE hArr[5] = { INVALID\_HANDLE\_VALUE };

for (int i = 0; i < 5; ++i)

{

Car \*pCar = new Car;

sprintf(pCar->name, "车辆%c", 'A' + i);

pCar->time = 3 + i \* 3;

//创建车辆线程

hArr[i]= CreateThread(NULL, 0, ThreadFun, (LPVOID)pCar, 0, NULL);

}

//等待所有线程执行完毕

WaitForMultipleObjects(5, hArr, true, INFINITE);

return 0;

}

DWORD WINAPI ThreadFun(LPVOID paramter)

{

//如果有剩余停车位资源（有信号状态），就放行

WaitForSingleObject(hSemaphore, INFINITE);

Car \*pCar = (Car \*)paramter;

printf("%s进入停车场,停车%d秒！\n", pCar->name, pCar->time);

Sleep(pCar->time \* 1000);

printf("%s驶离停车场！\n", pCar->name);

//释放一个停车位（信号量+1）

ReleaseSemaphore(hSemaphore,1,NULL);

return 0;

}

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

**6. 利用Semaphore实现程序只允许一个进程运行**

#include<windows.h>

#include<stdio.h>

int main()

{

//利用Semaphore实现程序只允许一个进程运行

HANDLE hSemaphore = CreateSemaphore(NULL, 3, 3, L"停车位");

if(GetLastError()== ERROR\_ALREADY\_EXISTS)

{

printf("程序已经允许，请不要开启多个进程！");

getchar();

CloseHandle(hSemaphore);

return 0;

}

printf("程序第一次启动！");

getchar();

return 0;

}