Semaphore

1 Khái niệm

Semaphore là một đối tượng đồng bộ hóa lưu giữ một biến đếm có giá trị từ 0 đến Max, semaphore nhận trạng thái TRUE khi giá trị của biến đếm >0, và nhận trạng thái FALSE nếu có giá trị của biến đếm = 0.

2 Tình huống sử dụng

Đối tượng Semaphore được dùng để kiểm soát việc cho phép một số hữu hạn tiểu trình cùng lúc truy xuất một tài nguyên dùng chung thông qua biến đếm count. Đối tượng đồng bộ Semaphore là đối tượng trong có một biến đếm count mang giá trị từ 0 đến giá trị max (được xác định khi tạo đối tượng). Biến count sẽ giảm một đơn vị khi có một tiểu trình sử dụng đối tượng này để đồng bộ (khi tiểu trình gọi **wait functions** với handle của đối tượng này) và tăng một đơn vị mỗi lần một tiểu trình gọi hàm ReleaseSemaphore.

Đối tượng Semaphore ở trạng thái nonsignal khi biến count = 0, nghĩa là có max tiểu trình đang truy xuất tài nguyên chung đó. Và ở trạng thái signal khi biến count > 0 .

Tiểu trình sử dụng hàm CreateSemaphore để tạo một đối tượng semaphore và dùng **wait functions** để chờ. Nếu các tiểu trình rơi vào trạng thái đợi, đối tượng semaphore sử dụng chiến lược FIFO để giải phóng các tiểu trình chờ rất hợp lý: tiểu trình nào hàng đợi đầu tiên sẽ được giải phóng sớm nhất bất kể độ ưu tiên của tiểu trình này thấp hơn các tiểu trình vào sau.

3 Các bước thiết lập và sử dụng Semaphore

Sử dụng Semaphore trong việc đồng bộ hoá **các tiểu trình trong một hay nhiều tiến trình** cần tiến hành theo các bước sau:

	Sử dụng các hàm API	**Sử dụng lớp do MFC hỗ trợ**
1. Tạo lập đối tượng Semaphore	Tiểu trình gọi hàm CreateSemaphore() để tạo lập và trả về handle của đối tượng semaphore, hoặc gọi hàm OpenSemaphore() để lấy handle của một đối tượng semaphore đã tồn tại khi biết tên đối tượng.	CSemaphore < đối tượng S >
2. Đăng ký truy xuất tài nguyên	Trong các tiểu trình, tại vị trí bắt đầu vào đoạn lệnh truy xuất đến tài nguyên dùng chung cần được bảo vệ, gọi hàm WaitForSingleObject() để đăng ký truy xuất tài nguyên tuỳ vào trạng thái của semaphore. Nếu	< đối tượng S >.Lock()

	đối tượng này ở trạng thái signal, nghĩa là tài nguyên Semaphore bảo vệ còn chấp nhận cho tiểu trình truy xuất, khi đó hàm WaitForSingleObject() sẽ giảm biến count của semaphore đi 1 đơn vị. Ngược lại, nếu Semaphore đang ở trạng thái nonsignal, nghĩa là không thể chấp nhận thêm một tiểu trình nào khác truy xuất đến tài nguyên đó, thì hàm WaitForSingleObject() không thể trả về và tiểu trình này sẽ chuyển sang trạng thái chờ cho đến khi có một tiểu trình khác chấm dứt truy xuất tài nguyên và làm biến count của Semaphore tăng lên.	
3. Tăng giá trị biến đếm của Semaphore	Trong các tiểu trình , tại vị trí kết thúc đoạn lệnh truy xuất đến tài nguyên dùng chung đang được bảo vệ , gọi hàm ReleaseSemaphore() để báo hiệu cho hệ thống biết tiểu trình đã kết thúc việc truy xuất tài nguyên và làm cho biến đếm count của Semaphore > 0, do đó trạng thái của Semaphore nhận trạng thái signal, tạo cơ hội cho các tiểu trình khác có thể truy xuất tài nguyên chung được Semaphore bảo vệ.	< đối tượng S >.UnLock()
4. Kết thúc sử dụng đối tượng Semaphore	Khi không còn nhu cầu truy xuất tài nguyên chung được bảo vệ bởi một đối tượng Semaphore, tiểu trình gọi hàm CloseHandle() để kết thúc sử dụng đối tượng Semaphore tương ứng.	

Lưu ý: Các bước 1, 4 thường được tiểu trình chính thực hiện, và các bước còn lại 2, 3 do các tiểu trình cần đồng bộ hoá thực hiện.

3.1 Ví dụ sử dụng Semaphore cho độc quyền truy xuất

3.1.1 Sử dụng các hàm API

```
...
HANDLE hsemaphoreX; // biến semaphore bảo vệ cho tài nguyên X
...
```

```
...MyThread1(...)
{
        ...
        WaitForSingleObject(hsemaphoreX, INFINITE); // vào miền găng
                                                    // miền_găng_sử_dụng_tài_nguyên_X();
        ReleaseSemaphore(hsemaphoreX, 1, NULL); // ra khỏi miền găng
        ...
}

... MyThread2(...)
{
        ...
        WaitForSingleObject(hsemaphoreX, INFINITE); // vào miền găng
                                                    // miền_găng_sử_dụng_tài_nguyên_X();
        ReleaseSemaphore(hsemaphoreX, 1, NULL); // ra khỏi miền găng
         ...
}

... main(...)
{
        hsemaphoreX = CreateSemaphore(NULL, 1, 1, NULL); // tạo biến semaphore
                                                         // khởi tạo e(s)=1, max(s)=1

         ....
}
```

3.1.2 Sử dụng lớp MFC hỗ trợ

```
...
CSemaphore semaphoreX; // biến semaphore bảo vệ cho tài nguyên X
...

... MyThread1(...)
{
        ...
        semaphoreX.Lock(); // vào miền găng miền_găng_sử_dụng_tài_nguyên_X();
        semaphoreX.Unlock(); // ra khỏi miền găng
        ...
}

... MyThread2(...)
{
        ...
        semaphoreX.Lock(); // vào miền găng miền_găng_sử_dụng_tài_nguyên_X();
        semaphoreX.Unlock(); // ra khỏi miền găng
        ...
}
```

3.2 Ví dụ sử dụng Semaphore cho phối hợp hoạt động

3.2.1 Sử dụng các hàm API

```
...
HANDLE hsemaphoreX; // biến semaphore đồng bộ hóa sự kiện X
...

... ProducerThread(...)
{
        while (...)
        {
                ...
                produceX();
                ReleaseSemaphore(hsemaphoreX, 1, NULL); // báo hiệu đã sẵn sàng
                ...
        }
}

... ConsumerThread(...)
{
        while (...)
        {
                ...
                WaitForSingleObject(hsemaphoreX, INFINITE); // đợi tín hiệu sẵn sàng
                consumeX();
                ...
        }
}

... main(...)
{
        hsemaphoreX=CreateSemaphore(NULL,0,0x7FFFFFFF,NULL); // tạo
                                                    semaphore e(s)=0, max(s)=maxint
        ....
}
```

3.2.2 Sử dụng lớp MFC hỗ trợ

```
...
CSemaphore semaphoreX(0, 0x7FFFFFFF); // biến semaphore đồng bộ hóa sự kiện X
...

... ProducerThread(...)
{
        while (...)
        {
                ...
                produceX();
                semaphoreX.Unlock(); // báo hiệu đã sẵn sàng
```

```
                ...
        }
}

... ConsumerThread(...)
{
        while (...)
        {
                ...
                semaphoreX.Lock(); // đợi tín hiệu sẵn sàng
                ...
        }
}
```

4 Các hàm liên quan

```
HANDLE    CreateSemaphore(
              LPSECURITY_ATTRIBUTE  lpSemaphoreAttributes,
              LONG                  lInitialCount,
              LONG                  lMaximumCount,
              LPCTSTR               lpName
);
```

Hàm sẽ tạo 1 đối tượng Semaphore và trả về handle của đối tượng này.

- ***lpSemaphoreAttributes*** : sẽ cài đặt thuộc tính bảo mật cho đối tượng Semaphore, nếu là NULL thì sẽ lấy thuộc tính bảo mật mặc định.
- ***lInitialCount*** : chỉ định giá trị khởi tạo cho biến count trong đối tượng Semaphor từ 0 cho đến lMaximumCount.
- ***lMaximumCount*** : cho biết số tiểu trình tối đa có thể đồng thời truy xuất tài nguyên.
- ***lpName*** : khi muốn chia sẻ đối tượng semaphore cho các tiểu trình giữa các tiến trình khác nhau để cùng bảo vệ 1 tài nguyên chung nào đó thì ta cần đặt 1 tên cho đối tượng semaphore. Nếu không thì đặt tham số này là NULL.

```
HANDLE    OpenSemaphore(
              DWORD     dwDesiredAccess,
              BOOL      bInheritHandle,
              LPCTSTR   lpName
);
```

Gọi 1 đối tượng Semaphore đã tồn tại và hàm sẽ trả về handle của đối tượng Semaphore đó. Khi 1 tiến trình tạo 1 đối tượng với 1 tên xác định thì tiến trình khác sẽ gọi hàm này với tên Semaphore để nhận được handle của đối tượng Semaphore.

- ***dwDesiredAccess*** : cho biết chế độ truy xuất, có kể kết hợp của các giá trị cờ sau: SEMAPHOR_ALL_ACCESS , SEMAPHORE_MODIFY_STATE, hay SYNCHRONIZE.
- ***bInheritHandle*** : cho biết giá trị handle trả về có được kế thừa hay không. Nếu là TRUE thì các tiến trình tạo bởi hàm CreateProcess có thể kế thừa, ngược lại thì không kế thừa .
- ***lpName*** :Tên của đối tượng Semaphore đã tồn tại cần lấy handle.

```
BOOL    ReleaseSemaphore(
            HANDLE    hSemaphore,
            LONG      lReleaseCount,
            LPLONG    lpPreviousCount
);
```

Khi 1 tiểu trình ngưng việc sử dụng tài nguyên sẽ gọi đến hàm này để tăng giá trị của biến đếm của đối tượng semaphore lên lReleaseCount .Giá trị của lReleaseCount >0 và sau khi tăng phải đảm bảo rằng biến đếm <= maxvalue.

- ***hSemaphore*** : handle của đối tượng semaphore.
- ***lReleaseCount*** : giá trị mà mỗi lần gọi thì biến đếm của đối tượng semaphore được tăng lên, thông thường là 1.
- ***lpPreviousCount*** : nhận giá trị trả về của biến đếm trước khi tăng lên, nếu không quan tâm giá trị này thì gán là NULL.

5 Ví dụ

Chương trình **TestSemaphore** là một ứng dụng gồm 2 tiểu trình:

- 1 tiểu trình chính: có chức năng ghi giá trị của biến toàn cục Count ra file
- 1 tiểu trình phụ: có chức năng tăng giá trị của biến toàn cục Count lên n=100 lần

Nếu không sử dụng Semaphore (không chọn chức năng Use Semaphore) thì kết quả nhận được của biến Count trong file output sẽ không tăng nghiêm ngặt.

Nếu sử dụng Semaphore (chọn chức năng Use Semaphore) thì kết quả nhận được của biến Count trong file output sẽ tăng nghiêm ngặt.