インターシステムズ製品のセマフォ

Version 2024.1
2024-06-03

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

インターシステムズ製品のセマフォ
InterSystems IRIS Data Platform　Version 2024.1　2024-06-03

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

インターシステムズ製品のセマフォ

1 背景

Wikipedia でのセマフォの定義は次のとおりです。コンピュータ・サイエンスとりわけオペレーティング・システムでは、セマフォは、並列プログラミングまたは複数ユーザ環境において複数のプロセスによって一般的なリソースへのアクセスの制御に使用される変数または抽象的なデータ型です。*セマフォは、ミューテックス（またはロック）とは異なります。ミューテックスは、単一のリソースにアクセスする競合プロセスを規制するために最も頻繁に使用されます。セマフォは、同一のリソースのコピーが複数あり、これらのコピーそれぞれを独立したプロセスで同時に使用できる場合に使用されます。

事務用品店について考えてみます。顧客が使用するためのコピー機が数台あるが、各コピー機は同時に 1 人の顧客しか使用できません。これを制御するために、コピー機の使用を可能にし、使用量を記録する一連の鍵があります。顧客は、ドキュメントのコピーを希望する場合、店員に鍵を要求してコピー機を使用し、鍵を返し、使用量に対して支払いを行います。すべてのコピー機が使用中の場合、顧客は鍵が返されるまで待つ必要があります。鍵が保管される場所がセマフォとして機能します。

この例は、コピーのサイズによって区別するなどさまざまなタイプのコピー機を含むようにさらに一般化できます。この場合、複数のセマフォがあり、コピー機がコピーのサイズにおいて重複する場合、この共通のサイズのコピーが必要な顧客は 2 つのリソースを使用できるようになります。

2 概要

InterSystems IRIS® でのセマフォは、プロセス間で高速かつ効率的な通信を提供するために使用される共有オブジェクトです。各セマフォは %SYSTEM.Semaphore クラスのインスタンスです。セマフォは、64 ビットの負でない整数を保持する共有変数としてモデル化できます。セマフォの操作は、この変数を共有するすべてのプロセスで同期化された方法で変数の値を変更します。規約では、値の変更によって、セマフォを共有するプロセス間に情報が通知されます。

セマフォとロックには共通点が多いように思えますが、セマフォの使用にはいくつかの利点があります。最大の利点は、セマフォが付与されたときに通知が送信されることです。したがって、セマフォを使用するプロセスは、プロセッサ・サイクルを消費したり、アプリケーション・ロジックを複雑化してロックが解放されたかどうかを確認するためにロックをポーリングしたりする必要がありません。さらに、セマフォは ECP 接続に対して透過的に動作し、同じ状況でロックを確認するために必要な交換よりもはるかに効率的です。

3 セマフォの概要

3.1 セマフォの名前

セマフォは、作成時に ObjectScript 文字列として提供される名前によって識別されます。セマフォに与えられる名前は、ローカル変数およびグローバル変数の規則に準拠することが期待されています。セマフォの名前は、他のすべての既存のセマフォから区別するためにのみあります。有効な名前文字列の例は、`SlotsOpen`、`J(3)`、`^pendingRequest(""j"")` などです。

通常セマフォは、そのセマフォが作成されたインスタンスに格納され、そのインスタンスのすべてのプロセスで認識できます。ただし、セマフォの名前がグローバル変数の名前のように見える場合、そのセマフォは、グローバル変数（添え字を含む）がマップされるシステムに格納されます。このことによって、そのようなセマフォは、ECP システムのインスタンスで実行されているすべてのプロセスで認識できるようになります。

注釈　セマフォの名前は文字列ですが、実行時には `"^" _ BaseName _ "(" _ (1 + 2) _ ")"` のような式から構築できます。BaseName に文字列“Printers”が含まれている場合、そのセマフォの名前は `^Printers(3)` になります。

3.2 セマフォの値

セマフォの値は、63 ビットの符号なし整数として格納されるため、常に 0 以上です。

最大の 63 ビットの符号なし整数は 9,223,372,036,854,775,807 ((2 ** 63) - 1) です。セマフォは、この値を超えてインクリメントできません。また、0 未満にデクリメントすることもできません。

3.3 セマフォのインスタンスと変数

セマフォは、%SYSTEM.Semaphore から派生したクラスのインスタンスです。セマフォが作成されて初期化されると、通常その OREF は、他の操作で使用できるように ObjectScript 変数に格納され、引数として渡され、最終的に削除されます。セマフォへの参照を含む変数の名前はセマフォの名前と対応する必要はありませんが、プログラミング手法として関係性を持たせることをお勧めします。

非永続データへのあらゆるオブジェクト参照のように、セマフォへの最後の参照が再要求されると、基本のセマフォも削除されます。

3.4 セマフォ操作

3.4.1 基本

セマフォ操作は、2 つの大きなグループに分割できます。セマフォを直接操作するものと、セマフォを操作するために他のプロセスを待機するものです。最初のグループには以下のものが含まれます。

- Create – 新しいセマフォのインスタンスを作成し、使用できるように初期化します。
- Open – 既存のセマフォにアクセスし、初期化します。
- Delete – セマフォを認識しているあらゆるプロセスがそのセマフォを使用できないようにします。
- Increment – 指定の数をセマフォの値に追加します。
- Decrement – この操作は、セマフォの値が 0 の場合、値が正の値になるまで待ちます。セマフォが正の値の場合、デクリメント量（またはセマフォの値のうちどちらか小さい方）をセマフォから減算します。
- GetValue – セマフォの現在値を返します。
- SetValue – 負でない指定整数にセマフォの現在値を設定します。

3.4.2 複数のセマフォの管理

操作の 2 番目のグループは、セマフォのリストをグループとして管理し、それぞれに対して保留されている操作が完了するのを待ちます。

- AddToWaitMany – 指定されたセマフォ操作を待機リストに追加します。
- RemoveFromWaitMany – 指定されたセマフォ操作を待機リストから削除します。

- WaitMany – 待機リスト上のすべてのセマフォが個々の操作を完了するのを待ちます。この操作はタイムアウトになる可能性があります。

3.4.3 待機リスト

WaitMany を使用して複数のセマフォを調整する各プロセスは、内部リストでセマフォのデクリメント操作要求を保持します。リスト上の操作は以下のように処理されます。

- AddToWaitMany メソッドが呼び出されてリストにデクリメント操作が配置されると、システムはそのときにデクリメントを実行しようとします。セマフォの値が 0 以外の場合、デクリメントは成功します。デクリメントされる量は、セマフォの値と要求された量の小さい方です。セマフォの値より大きい要求量は無視されます。
- 要求がリストに追加された時点でセマフォの値が 0 の場合、何も行われず、要求は保留と見なされます。将来のある時点で、対象のセマフォが 0 以外の値になった場合、InterSystems IRIS は、そのセマフォを参照する操作を含むプロセスの 1 つを選択し、デクリメント操作を実行します。その操作の結果、セマフォの値が依然として 0 以外の場合、InterSystems IRIS は、要求がなくなるかまたはセマフォの値が 0 になるまでプロセスを繰り返します。
- プロセスが WaitMany メソッドを呼び出すと、InterSystems IRIS は、待機リストの操作のそれぞれを調べます。満たされた要求を探し、InterSystems IRIS は、対象のセマフォの WaitCompleted メソッドを呼び出し、その要求を待機リストから削除します。満たされた要求の処理を完了したら、その要求の数を呼び出し元に返します。待機タイムアウトを超えた場合 0 を返します。保留中の要求と満たされていない要求は待機リストに残ります。
- セマフォの非同期的な特性のために、待機リスト上の保留中の要求の 1 つが WaitManyの呼び出し中に満たされる場合があります。現在満たされているこの要求が WaitManyの今回の呼び出し中にカウントされるか、次の呼び出しでカウントされるかは不定です。
- セマフォへの OREF を保持しているプロセスがこれを削除する可能性もあります。この場合、何が起こるかは要求された操作が満たされているかどうかで決まります。
  - 満たされた操作の対象がこのセマフォである場合、この要求はセマフォを 0 だけデクリメントしたとしてマークされます。セマフォが存在しないため WaitCompleted メソッドを呼び出すことはできませんが、この要求は、WaitManyが返す値で満たされたとしてカウントされます。
  - 要求がまだ保留中の場合、単純に待機リストから削除されます。

3.4.4 コールバック

セマフォのインスタンスは、ユーザが実装することを期待されている抽象メソッド（WaitCompleted）を継承します。待機リスト上の満たされた要求を処理すると、WaitMany は待機リストの各セマフォに対して、セマフォがデクリメントされた量を引数として渡して WaitCompleted メソッドを呼び出します。WaitCompleted が復帰すると、WaitMany は待機リストから要求を削除します。

3.4.5 その他の考慮事項

同じ待機リスト上の複数のデクリメント要求

同じ待機リストで同じセマフォに複数回デクリメントを要求することはエラーではありません。追加された要求は以下のように処理されます。

- 最初の要求が満たされていない場合、2 番目の要求のデクリメント量は最初の要求のデクリメント量に追加されます。
- 最初の要求が（完全または部分的に）満たされた場合、2 番目の要求は通常どおりに処理されます。つまり、セマフォの値が 0 以外の場合、デクリメント要求は完全または部分的に満たされます。ただし、実際にデクリメントされる量は、最初の要求によって取得された量に追加されます。

デクリメント量は、WaitMany が呼び出されるまでコールバックによって報告されません。したがって、同じセマフォに対する複数の要求は、1 つの結合された要求が行われたかのように見えます。この結果を以下のシナリオに示します。

1. セマフォ A を 0 に設定します。
2. A に対して 4 のデクリメント要求をします。
3. A に対して 1 のデクリメント要求をします。新しいデクリメント = 5 になります。
4. セマフォ A を 4 に設定します。
5. 要求が満たされます。4 が付与されます。
6. WaitMany の呼び出しが行われます。A に対する WaitCompleted によって 4 が報告されます。

1. セマフォ A を 1 に設定します。
2. A に対して 3 のデクリメント要求をします。
3. 要求が満たされます。1 が付与されます。
4. A に対して 4 のデクリメント要求をします。
5. WaitMany の呼び出しが行われます。A に対する WaitCompleted によって 1 が報告されます。

1. セマフォ A を 1 に設定します。
2. A に対して 3 のデクリメント要求をします。
3. 要求が満たされます。1 が付与されます。
4. A に対して 4 のデクリメント要求をします。
5. セマフォ A を 5 に設定します。
6. 要求が満たされます。4 が付与されます。7.WaitMany の呼び出しが行われます。A に対する WaitCompleted によって 5 が報告されます。セマフォ A の値 = 1 になります。

セマフォの削除

セマフォには所有者はいません。また、セマフォはオブジェクト・インスタンスのように参照カウントされません。セマフォを開くことができるプロセスならばどれでもセマフォを削除できます。

セマフォが削除されると以下のようになります。

- 該当のセマフォの保留中のデクリメントがいずれかの待機リストにある場合、WaitCompleted コールバックがデクリメント値 0 で呼び出されます。
- 該当のセマフォは、マップされているシステム (ローカルまたはリモート) から削除されます。
- 他のプロセスがさらにアクセスしようとすると、<INVALID SEMAPHORE> エラーで失敗します。

ジョブ終了と待機リスト

InterSystems IRIS プロセスが終了すると、その待機リストは解放されます。待機リストに残っているが WaitMany によって処理されなかった満たされたデクリメント要求はクリアされます。個々のデクリメント量は、デクリメントされたセマフォに戻されて追加されません。待機リストの満たされていない要求は単純に削除されます。

セマフォと ECP

ECP システム上のセマフォの場合、セマフォの操作は、そのセマフォを保持しているシステムに要求が到着した順序に従って並べられます。各操作は、必ず次の操作が開始される前に完了します。リモート・セマフォの場合、以下の条件が保証されます。

- セマフォのインクリメントとデクリメントは SET および KILL の後に発生します。
- セマフォが SET、インクリメントまたはデクリメントされるとき、ECP データ・キャッシュは、サーバで後続の SET、インクリメント、またはデクリメントを基準に整合をとられます。

セマフォは永続的ではないため、サービス中断が発生すると、ECP システム上でサーバをまたがる保留中のセマフォ操作はリカバリできません。サーバまたはネットワークの障害による ECP 停止の後、アプリケーション・サーバのセマフォは削除され、データ・サーバ上の保留中の要求は削除されます。この状況を検出して正しい状態で必要なセマフォを再作成する処理は、アプリケーションで行う必要があります。

4 単純なプロデューサ/コンシューマの例

セマフォを使用したプロデューサ/コンシューマのシナリオを実装する一連のクラスを以下に示します。“主な“ プロセスは、セマフォを初期化し、アクティビティがすべて終了したことをユーザが示すことを待ちます。プロデューサは、更新間に変数を遅延させてセマフォ値をループでランダムにインクリメントします。コンシューマは、同様にループ内でランダムなタイミングでランダムの量をセマフォから削除しようとします。この例は 5 つのクラスから構成されます。

- Main – 環境を初期化し、セマフォのアクティビティが完了するのを待機するクラス。
- Counter – セマフォ自体を実装するクラス。これは、セマフォが作成されセマフォが待機リストに配置された結果として発生するコールバックをログに記録します。
- Producer – メイン・メソッドがセマフォの値をインクリメントするクラス。インクリメント量は、ランダムに選択される小さい整数です。インクリメントを行った後、このメソッドは、次のインクリメントの前にランダムな秒の時間 (数秒) 遅延します。
- Consumer – これは Producer を補完するものです。このクラスのメイン・メソッドは、ランダムに選択される小さい整数だけセマフォをデクリメントしようとします。このデクリメント要求を待機リストに追加しますが、その際、これもランダムに選択される秒数である待機時間を伴います。
- Util – このクラスには、この例の他のクラスで使用されるいくつかのメソッドがあります。いくつかのメソッドは、すべてのアクティビティに関する共通のログを管理する際の問題に対処します。その他のメソッドは、複数のコンシューマおよび複数のプロデューサの命名を処理します。

個々のクラスの本文をこれらの動作のより詳細な検討と共に以下に示します。その後、クラスの実行方法の説明および可能なバリエーションがあります。

注釈　クラスを構成するコードは、意図的に単純化して記述しています。できる限り、各文は 1 つのアクションのみを実行しています。ユーザによる例の変更が簡単になり、わかりやすくなっているはずです。

4.1 クラス : Semaphore.Main

このクラスはデモ環境を確立します。ユーティリティ・クラスを呼び出し、ログおよび名前の索引作成機能を初期化します。次に、初期値 0 で一般的なセマフォを初期化し、デモの終了を指示する (通常 ENTER キー) をユーザが入力するのを待ちます。

ユーザ入力を受け取ると、セマフォの現在値を報告して削除しようとし、実行を終了します。

Class Definition

```
/// Environment setup for example
Class Semaphore.Main Extends %RegisteredObject [ ProcedureBlock ]
{

/// The name of the shared semaphore
Parameter ME = "Main";

/// driver for the semaphore demo
ClassMethod Run()
{
    // initialize the logging globals
    Do ##class(Semaphore.Util).InitLog()
    Do ##class(Semaphore.Util).InitIndex()
```

```
    Set msg = ..#ME _ " Started"
    Do ..Log(msg)

    // create and initialize the semaphore
    Set inventory = ##class(Semaphore.Counter).%New()
    If ('($ISOBJECT(inventory))) {
        Set msg = "%New() of MySem failed"
        Do ..Log(msg)
        Quit
    }

    Set msg = "Semaphore create result: " _ inventory.Init(0)
    Do ..Log(msg)

    // wait for termination response
    Set msg = "Enter any character to terminate Run method"
    Do ..Log(msg)

    Read *x

    // report final value, remove the semaphore and finish
    Set msg = "Final value = " _ inventory.GetValue()
    Do ..Log(msg)
    Set msg = "Semaphore delete status: " _ inventory.Delete()
    Do ..Log(msg)
    Set msg = ..#ME _ " Finished"
    Do ..Log(msg)

    Quit
}

/// Enter messages as received into a common log
ClassMethod Log(msg As %String) [ Private ]
{
    Do ##class(Semaphore.Util).Logger(..#ME, msg)
    Quit
}

}
```

4.2 クラス : Semaphore.Counter

このクラスは、この例で使用するセマフォを実装します。必要に応じて、これは %SYSTEM.Semaphore のサブクラスで、メソッド WaitCompleted の実装を提供します。簡略化のために、セマフォを初期化するコードもこのクラスに含まれます。このセマフォの名前を提供し、設定、プロデューサ、およびコンシューマの各クラスによる取得を可能にするクラス・メソッドもあります。

Class Definition

```
/// Local semaphore example class
Class Semaphore.Counter Extends %SYSTEM.Semaphore
{

/// Direct messages to the log facility
ClassMethod Name() As %String
{
    Quit "Counter"
}

/// Direct messages to the log facility
Method Log(Msg As %String) [ Private ]
{
    Do ##class(Semaphore.Util).Logger(..Name(), Msg)
    Quit
}

/// Return the semaphore id value as a hex string
Method MyId() As %String
{
   Quit ("0x" _ $ZHEX(..SemID))
}

/// Invoked when instance created
Method %OnNew() As %Status
{
    Set msg = "New semaphore"
    Do ..Log(msg)
```

```
    Quit $$$OK
}

Method Init(initvalue = 0) As %Status
{
    Try {
        If (..Create(..Name(), initvalue)) {
            Set msg = "Created: """ _ ..Name()
                    _ """; Value = " _ initvalue
                    _ "; Id = 0x" _ ..MyId()
            Do ..Log(msg)
            Return 1
        }
        Else {
        Set msg = "Semaphore create failed: Name = """ _ ..Name() _ """"
        Do ..Log(msg)
        Return 0
        }
    } Catch {
        Set msg = "Semaphore failure caught"
        Do ..Log(msg)
        Return 0
    }
}

Method %OnClose() As %Status [ Private ]
{
    Set msg = "Closing Semaphore: Id = " _ ..MyId()
    Do ..Log(msg)
    Quit $$$OK
}

/// This method is invoked by WaitMany() as a callback.
/// Either a non-zero amount was available in the semaphore or the wait timed out.
/// The amount decremented is passed as the argument to this method; zero, in the
/// case of a timeout.
///
/// After invoking this method, the semaphore is removed from the wait many list.
/// An explicit invocation of AddToWaitMany is required to put it back
/// into the wait list.
Method WaitCompleted(amt As %Integer)
{
    // just report the decrement amount
    Set msg = "WaitCompleted: " _ ..MyId() _ "; Amt = " _ amt
    Do ..Log(msg)

    Quit
}

}
```

4.3 クラス : Semaphore.Producer

このクラスは、一般的なセマフォへの OREF を取得するためのものです。OREF を取得すると、ランダムに選択される小さい整数だけセマフォを繰り返しインクリメントしようとし、各インクリメント間にランダムに選択される小さい秒数の間、一時停止します。セマフォをインクリメントする各試行はログに入力されます。

Class Definition

```
/// The semaphore increment class
Class Semaphore.Producer Extends %RegisteredObject [ ProcedureBlock ]
{

/// My class name
Parameter MeBase = "Producer";

/// Increments the semaphore by small random amounts after pauses
ClassMethod Run() As %Status
{
    // establish name and access semaphore
    Set ME = ##class(Semaphore.Util).IndexName(..#MeBase)
    Set msg = ME _ " Started"
    Do ..Logger(ME, msg)

    Set cell = ##class(Semaphore.Counter).%New()
    Do cell.Open(##class(Semaphore.Counter).Name())

    Set msg = "Open Id = " _ cell.MyId()
    Do ..Logger(ME, msg)
```

```
    // increment semaphore by random amounts
    // at random times
    For addcnt = 1 : 1 : 8 {
        Set incamt = $RANDOM(5) + 1
        Set waitsec = $RANDOM(10) + 1
        Set msg = "Increment " _ cell.MyId()
                _ " = " _ cell.GetValue()
                _ " by " _ incamt
                _ " wait " _ waitsec _ " sec"
        Do cell.Increment(incamt)
        Do ..Logger(ME, msg)
        Hang waitsec
    }

    // finish
    Set msg = ME _ " Finished"
    Do ..Logger(ME, msg)

    Quit $$$OK
}

/// Channels messages to the central logger
ClassMethod Logger(id As %String, msg As %String) [ Private ]
{
    Do ##class(Semaphore.Util).Logger(id, msg)
    Quit
}

}
```

4.4 クラス : Semaphore.Consumer

このクラスは Semaphore.Producer を補完するものです。これも Producer と同様の方法で一般的なセマフォへの OREF を取得し、ランダムに選択される量だけセマフォを繰り返しデクリメントしようとし、各試行の間にランダムに選択される一時停止を行います。各試行の成功または失敗はログに書き込まれます。

Class Definition

```
/// The semaphore decrement class
Class Semaphore.Consumer Extends %RegisteredObject [ ProcedureBlock ]
{

/// My class name
Parameter MeBase = "Consumer";

/// Decrements the semaphore by small random amounts after pauses
ClassMethod Run() As %Status
{
    // establish name and access semaphore
    Set ME = ##class(Semaphore.Util).IndexName(..#MeBase)
    Set msg = ME _ " Started"
    Do ..Logger(ME, msg)

    Set cell = ##class(Semaphore.Counter).%New()
    Do cell.Open(##class(Semaphore.Counter).Name())
    Set msg = "Consumer: Open Id = " _ cell.MyId()
    Do ..Logger(ME, msg)

    // repeatedly decrement the semaphore by
    // variable amounts and varied times
    For deccnt = 1 : 1 : 15 {
        Set decamt = $RANDOM(5) + 1
        Set waitsec = $RANDOM(10) + 1
        Set msg = "Decrement " _ cell.MyId()
                _ " = " _ cell.GetValue()
                _ " by " _ decamt
                _ " wait " _ waitsec _ " sec"
        // in this case we wait for a single semaphore
        // but we could wait on multiple semaphore decrements
        // (up to 200) at one time
        Do cell.AddToWaitMany(decamt)
        Do ..Logger(ME, msg)
        Set result = ##class(%SYSTEM.Semaphore).WaitMany(waitsec)
        Set msg = $SELECT((result > 0):"Granted", 1:"Timeout")
        Do ..Logger(ME, msg)
    }

    // finish
```

```
    Set msg = ME _ " Finished"
    Do ..Logger(ME, msg)

    Quit $$$OK
}

/// Channels messages to the central logger
ClassMethod Logger(id As %String, msg As %String) [ Private ]
{
    Do ##class(Semaphore.Util).Logger(id, msg)
    Quit
}

}
```

4.5 クラス : Semaphore.Util

このクラスには、この例に関する 2 つの問題に対処するメソッドが含まれています。1 つ目は、ログされるメッセージおよびログに送信されるメッセージとその後の表示の両方をアーカイブ処理するメソッドを保持するために必要な構造体の初期化です。

2 つ目の一連のメソッドは、プロデューサとコンシューマを識別するための番号付けされた順序の名前の生成を処理します。これは厳密には必要ありません。その理由は $JOB コマンドによって提供される InterSystems IRIS のプロセス ID でも行われるためです。ただし、読みやすいラベルを使用する方が簡単です。

Class Definition

```
/// Utility class
Class Semaphore.Util Extends %RegisteredObject [ ProcedureBlock ]
{

/// The name of the shared semaphore
Parameter ME = "Util";

/// initialize output log
ClassMethod InitLog()
{
    // initialize the logging global
    Kill ^SemaphoreLog
    Set ^SemaphoreLog = 0

    Quit
}

/// Enter messages as received into a global
/// for logging purposes
ClassMethod Logger(sender As %String, msg As %String)
{
    Set inx = $INCREMENT(^SemaphoreLog)
    Set ^SemaphoreLog(inx, 0) = $JOB
    Set ^SemaphoreLog(inx, 1) = sender
    Set ^SemaphoreLog(inx, 2) = msg
    Write "(", ^SemaphoreLog, ") ", msg, !
    Quit
}

/// display the messages in the log
ClassMethod ShowLog()
{
    Set msgcnt = $GET(^SemaphoreLog, 0)
    Write "Message Log: Entries = ", msgcnt, !, !
    Write "#", ?5, "$JOB", ?12, "Sender", ?25, "Message", !

    For i = 1 : 1 : msgcnt {
        Set job = ^SemaphoreLog(i, 0)
        Set sender = ^SemaphoreLog(i, 1)
        Set msg = ^SemaphoreLog(i, 2)
        Write i, ")", ?5, job, ?12, sender, ":", ?25, msg, !
    }
    Quit
}

/// initialize the name index
ClassMethod InitIndex()
{
    // initialize the logging global
    Kill ^SemaphoreNames
```

```
    Quit
}

/// initialize the name index
ClassMethod IndexName(name As %String) As %String
{
    If ($DATA(^SemaphoreNames(name)) = 0) {
        Set ^SemaphoreNames(name) = 0
    }

    Set index =  $INCREMENT(^SemaphoreNames(name))
    Quit (name _ "." _ index)
}

}
```

4.6 例の実行

4.6.1 概要

Main、Producer、および Consumer の 3 つのクラスそれぞれには独自の Run メソッドがあり、それぞれ独自のターミナル・ウィンドウでそれぞれのメソッドを実行することが最適です。それぞれを実行すると、ログ用に生成されるメッセージが表示されます。待機している入力を行うことでユーザが Main クラスに応答すると、Main の Run メソッドはセマフォの削除を終了します。ユーザは、コマンドを入力することによってすべてのプロセスの統合されたログ・ファイルの表示を確認できます。

ObjectScript

```
Do ##class(Semaphore.Util).ShowLog()
```

注釈　以下の例はすべて、すべてのクラスが “USER“ ネームスペースでコンパイル済みであることを前提にしています。

4.6.2 例 1 – セマフォの作成と削除

この最も単純な例は、セマフォの作成と削除を示します。これは Semaphore.Main クラスを使用します。以下のことを実行します。

1. ターミナル・ウィンドウを開きます。
2. 次のコマンドを入力します。

   ```
   Do ##class(Semaphore.Main).Run()
   ```

3. このメソッドはセマフォを作成しようとします。正常に実行された場合は、“任意の文字を入力して Run メソッドを終了してください”というメッセージが表示されます。Enter キーを押します。このメソッドは、セマフォの初期化された値を表示し、これを削除して終了します。
4. 次のコマンドを発行してログ・ファイルを表示します。

   ```
   Do ##class(Semaphore.Util).ShowLog()
   ```

上記の手順に続いてターミナル・ウィンドウに表示されるメッセージ例を以下に示します。

```
USER>Do ##class(Semaphore.Main).Run()
(1) Main Started
(2) New semaphore
(3) Created: "Counter"; Value = 0; Id = 0x0x10000
(4) Semaphore create result: 1
(5) Enter any character to terminate Run method
<ENTER>
(6) Final value = 0
(7) Semaphore delete status: 1
(8) Main Finished
(9) Closing Semaphore: Id = 0x10000
```

ログ出力は次のようになります。

```
USER>Do ##class(Semaphore.Util).ShowLog()
Message Log: Entries = 9

#    $JOB   Sender       Message
1)   7176   Main:        Main Started
2)   7176   Counter:     New semaphore
3)   7176   Counter:     Created: "Counter"; Value = 0; Id = 0x0x10000
4)   7176   Main:        Semaphore create result: 1
5)   7176   Main:        Enter any character to terminate Run method
6)   7176   Main:        Final value = 0
7)   7176   Main:        Semaphore delete status: 1
8)   7176   Main:        Main Finished
9)   7176   Counter:     Closing Semaphore: Id = 0x10000
```

4.6.3 例 2 – セマフォを作成してからインクリメント

この例は、Producer の動作および両方のプロセスのログ・メッセージの取得を示します。

1. 2 つの別々のターミナル・ウィンドウを開きます。これらを“A”および“B”と呼びます。
2. ウィンドウ A で以下のコマンドを入力しますが、最後に ENTER キーを押さないでください。

   ```
   Do ##class(Semaphore.Main).Run()
   ```

3. ウィンドウ B で以下のコマンドを入力しますが、ここでもコマンドの最後に ENTER キーを押さないでください。

   ```
   Do ##class(Semaphore.Producer).Run()
   ```

4. ここで、ウィンドウ A で ENTER キーを押します。次にウィンドウ B で ENTER キーを押します。両方のクラスの並列実行が開始されますそれぞれの個々のメッセージがそれぞれのウィンドウに表示されます。
5. Producer プロセスが終了したら、B ウィンドウを閉じます。
6. A ウィンドウで、ENTER キーを押して、Main クラスを終了できるようにします。以下のコマンドでログを表示します。

   ```
   Do ##class(Semaphore.Util).ShowLog()
   ```

この例では、以下が出力されます。

ウィンドウ A

```
USER>Do ##class(Semaphore.Main).Run()
(1) Main Started
(2) New semaphore
(3) Created: "Counter"; Value = 0; Id = 0x0x30002
(4) Semaphore create result: 1
(5) Enter any character to terminate Run method
<ENTER>
(19) Final value = 28
(20) Semaphore delete status: 1
(21) Main Finished
(22) Closing Semaphore: Id = 0x30002
```

ウィンドウ B

```
USER>Do ##class(Semaphore.Producer).Run()
(6) Producer.1 Started
(7) New semaphore
(8) Open Id = 0x30002
(9) Increment 0x30002 = 0 by 5 wait 3 sec
(10) Increment 0x30002 = 5 by 2 wait 7 sec
(11) Increment 0x30002 = 7 by 2 wait 8 sec
(12) Increment 0x30002 = 9 by 4 wait 1 sec
(13) Increment 0x30002 = 13 by 5 wait 6 sec
(14) Increment 0x30002 = 18 by 3 wait 3 sec
(15) Increment 0x30002 = 21 by 4 wait 4 sec
(16) Increment 0x30002 = 25 by 3 wait 3 sec
(17) Producer.1 Finished
(18) Closing Semaphore: Id = 0x30002
```

ログ表示

```
USER>Do ##class(Semaphore.Util).ShowLog()
Message Log: Entries = 22

#    $JOB   Sender       Message
1)   7628   Main:        Main Started
2)   7628   Counter:     New semaphore
3)   7628   Counter:     Created: "Counter"; Value = 0; Id = 0x0x30002
4)   7628   Main:        Semaphore create result: 1
5)   7628   Main:        Enter any character to terminate Run method
6)   9036   Producer.1:  Producer.1 Started
7)   9036   Counter:     New semaphore
8)   9036   Producer.1:  Open Id = 0x30002
9)   9036   Producer.1:  Increment 0x30002 = 0 by 5 wait 3 sec
10)  9036   Producer.1:  Increment 0x30002 = 5 by 2 wait 7 sec
11)  9036   Producer.1:  Increment 0x30002 = 7 by 2 wait 8 sec
12)  9036   Producer.1:  Increment 0x30002 = 9 by 4 wait 1 sec
13)  9036   Producer.1:  Increment 0x30002 = 13 by 5 wait 6 sec
14)  9036   Producer.1:  Increment 0x30002 = 18 by 3 wait 3 sec
15)  9036   Producer.1:  Increment 0x30002 = 21 by 4 wait 4 sec
16)  9036   Producer.1:  Increment 0x30002 = 25 by 3 wait 3 sec
17)  9036   Producer.1:  Producer.1 Finished
18)  9036   Counter:     Closing Semaphore: Id = 0x30002
19)  7628   Main:        Final value = 28
20)  7628   Main:        Semaphore delete status: 1
21)  7628   Main:        Main Finished
22)  7628   Counter:     Closing Semaphore: Id = 0x30002
```

4.6.4 例 3 – 3 つのプロセスをすべて同時に実行

この例は、同じセマフォを一貫した方法でインクリメントおよびデクリメントしようとする試行を示します。3 つの主なクラスをすべて使用します。

1. 3 つの別々のターミナル・ウィンドウを開きます。これらを“A”、“B”、および“C”と呼びます。
2. ウィンドウ A で以下のコマンドを入力しますが、最後に ENTER キーを押さないでください。

   ```
   Do ##class(Semaphore.Main).Run()
   ```

3. ウィンドウ B で以下のコマンドを入力しますが、ここでもコマンドの最後に ENTER キーを押さないでください。

   ```
   Do ##class(Semaphore.Producer).Run()
   ```

4. ウィンドウ C で以下のコマンドを入力しますが、ここでもコマンドの最後に ENTER キーを押さないでください。

   ```
   Do ##class(Semaphore.Consumer).Run()
   ```

5. ウィンドウ A から各ウィンドウに移動して ENTER キーを押します。Main クラスが開始され、続いて他の 2 つのクラスが開始されます。前述のように、各プロセスはそれぞれ独自のウィンドウにそれぞれのログ・メッセージを表示します。
6. Producer プロセスと Consumer プロセスの両方が終了したら、B ウィンドウと C ウィンドウを閉じます。
7. A ウィンドウで、ENTER キーを押して、Main クラスを終了できるようにします。以下のコマンドでログを表示します。

   ```
   Do ##class(Semaphore.Util).ShowLog()
   ```

この例を実行すると、以下のような出力が生成されます。

ウィンドウ A

```
USER>Do ##class(Semaphore.Main).Run()
(1) Main Started
(2) New semaphore
(3) Created: "Counter"; Value = 0; Id = 0x0x60005
(4) Semaphore create result: 1
(5) Enter any character to terminate Run method
<ENTER>
(65) Final value = 0
(66) Semaphore delete status: 1
(67) Main Finished
(68) Closing Semaphore: Id = 0x60005
```

ウィンドウ B

```
USER>Do ##class(Semaphore.Producer).Run()
(6) Producer.1 Started
(7) New semaphore
(8) Open Id = 0x60005
(9) Increment 0x60005 = 0 by 2 wait 3 sec
(17) Increment 0x60005 = 0 by 5 wait 5 sec
(24) Increment 0x60005 = 0 by 4 wait 10 sec
(38) Increment 0x60005 = 0 by 2 wait 7 sec
(42) Increment 0x60005 = 0 by 1 wait 8 sec
(48) Increment 0x60005 = 0 by 1 wait 10 sec
(54) Increment 0x60005 = 0 by 5 wait 7 sec
(58) Increment 0x60005 = 0 by 1 wait 7 sec
(62) Producer.1 Finished
(63) Closing Semaphore: Id = 0x60005
```

ウィンドウ C

```
USER>Do ##class(Semaphore.Consumer).Run()
(10) Consumer.1 Started
(11) New semaphore
(12) Consumer: Open Id = 0x60005
(13) Decrement 0x60005 = 2 by 4 wait 1 sec
(14) WaitCompleted: 0x60005; Amt = 2
(15) Granted
(16) Decrement 0x60005 = 0 by 1 wait 4 sec
(18) WaitCompleted: 0x60005; Amt = 1
(19) Granted
(20) Decrement 0x60005 = 4 by 4 wait 6 sec
(21) WaitCompleted: 0x60005; Amt = 4
(22) Granted
(23) Decrement 0x60005 = 0 by 1 wait 9 sec
(25) WaitCompleted: 0x60005; Amt = 1
(26) Granted
(27) Decrement 0x60005 = 3 by 2 wait 4 sec
(28) WaitCompleted: 0x60005; Amt = 2
(29) Granted
(30) Decrement 0x60005 = 1 by 4 wait 4 sec
(31) WaitCompleted: 0x60005; Amt = 1
(32) Granted
(33) Decrement 0x60005 = 0 by 3 wait 1 sec
(34) Timeout
(35) Decrement 0x60005 = 0 by 1 wait 7 sec
(36) Timeout
(37) Decrement 0x60005 = 0 by 5 wait 7 sec
(39) WaitCompleted: 0x60005; Amt = 2
(40) Granted
(41) Decrement 0x60005 = 0 by 2 wait 8 sec
(43) WaitCompleted: 0x60005; Amt = 1
(44) Granted
(45) Decrement 0x60005 = 0 by 4 wait 2 sec
(46) Timeout
(47) Decrement 0x60005 = 0 by 4 wait 7 sec
(49) WaitCompleted: 0x60005; Amt = 1
(50) Granted
(51) Decrement 0x60005 = 0 by 5 wait 9 sec
(52) Timeout
(53) Decrement 0x60005 = 0 by 5 wait 9 sec
(55) WaitCompleted: 0x60005; Amt = 5
(56) Granted
(57) Decrement 0x60005 = 0 by 1 wait 9 sec
(59) WaitCompleted: 0x60005; Amt = 1
(60) Granted
(61) Consumer.1 Finished
```

ログ表示

```
USER>Do ##class(Semaphore.Util).ShowLog()
Message Log: Entries = 68

#    $JOB   Sender       Message
1)   3976   Main:        Main Started
2)   3976   Counter:     New semaphore
3)   3976   Counter:     Created: "Counter"; Value = 0; Id = 0x0x60005
4)   3976   Main:        Semaphore create result: 1
5)   3976   Main:        Enter any character to terminate Run method
6)   4852   Producer.1:  Producer.1 Started
7)   4852   Counter:     New semaphore
8)   4852   Producer.1:  Open Id = 0x60005
9)   4852   Producer.1:  Increment 0x60005 = 0 by 2 wait 3 sec
10)  6128   Consumer.1:  Consumer.1 Started
11)  6128   Counter:     New semaphore
12)  6128   Consumer.1:  Consumer: Open Id = 0x60005
13)  6128   Consumer.1:  Decrement 0x60005 = 2 by 4 wait 1 sec
14)  6128   Counter:     WaitCompleted: 0x60005; Amt = 2
15)  6128   Consumer.1:  Granted
16)  6128   Consumer.1:  Decrement 0x60005 = 0 by 1 wait 4 sec
17)  4852   Producer.1:  Increment 0x60005 = 0 by 5 wait 5 sec
18)  6128   Counter:     WaitCompleted: 0x60005; Amt = 1
19)  6128   Consumer.1:  Granted
20)  6128   Consumer.1:  Decrement 0x60005 = 4 by 4 wait 6 sec
21)  6128   Counter:     WaitCompleted: 0x60005; Amt = 4
22)  6128   Consumer.1:  Granted
23)  6128   Consumer.1:  Decrement 0x60005 = 0 by 1 wait 9 sec
24)  4852   Producer.1:  Increment 0x60005 = 0 by 4 wait 10 sec
25)  6128   Counter:     WaitCompleted: 0x60005; Amt = 1
26)  6128   Consumer.1:  Granted
27)  6128   Consumer.1:  Decrement 0x60005 = 3 by 2 wait 4 sec
28)  6128   Counter:     WaitCompleted: 0x60005; Amt = 2
29)  6128   Consumer.1:  Granted
30)  6128   Consumer.1:  Decrement 0x60005 = 1 by 4 wait 4 sec
31)  6128   Counter:     WaitCompleted: 0x60005; Amt = 1
32)  6128   Consumer.1:  Granted
33)  6128   Consumer.1:  Decrement 0x60005 = 0 by 3 wait 1 sec
34)  6128   Consumer.1:  Timeout
35)  6128   Consumer.1:  Decrement 0x60005 = 0 by 1 wait 7 sec
36)  6128   Consumer.1:  Timeout
37)  6128   Consumer.1:  Decrement 0x60005 = 0 by 5 wait 7 sec
38)  4852   Producer.1:  Increment 0x60005 = 0 by 2 wait 7 sec
39)  6128   Counter:     WaitCompleted: 0x60005; Amt = 2
40)  6128   Consumer.1:  Granted
41)  6128   Consumer.1:  Decrement 0x60005 = 0 by 2 wait 8 sec
42)  4852   Producer.1:  Increment 0x60005 = 0 by 1 wait 8 sec
43)  6128   Counter:     WaitCompleted: 0x60005; Amt = 1
44)  6128   Consumer.1:  Granted
45)  6128   Consumer.1:  Decrement 0x60005 = 0 by 4 wait 2 sec
46)  6128   Consumer.1:  Timeout
47)  6128   Consumer.1:  Decrement 0x60005 = 0 by 4 wait 7 sec
48)  4852   Producer.1:  Increment 0x60005 = 0 by 1 wait 10 sec
49)  6128   Counter:     WaitCompleted: 0x60005; Amt = 1
50)  6128   Consumer.1:  Granted
51)  6128   Consumer.1:  Decrement 0x60005 = 0 by 5 wait 9 sec
52)  6128   Consumer.1:  Timeout
53)  6128   Consumer.1:  Decrement 0x60005 = 0 by 5 wait 9 sec
54)  4852   Producer.1:  Increment 0x60005 = 0 by 5 wait 7 sec
55)  6128   Counter:     WaitCompleted: 0x60005; Amt = 5
56)  6128   Consumer.1:  Granted
57)  6128   Consumer.1:  Decrement 0x60005 = 0 by 1 wait 9 sec
58)  4852   Producer.1:  Increment 0x60005 = 0 by 1 wait 7 sec
59)  6128   Counter:     WaitCompleted: 0x60005; Amt = 1
60)  6128   Consumer.1:  Granted
61)  6128   Consumer.1:  Consumer.1 Finished
62)  4852   Producer.1:  Producer.1 Finished
63)  4852   Counter:     Closing Semaphore: Id = 0x60005
64)  6128   Counter:     Closing Semaphore: Id = 0x60005
65)  3976   Main:        Final value = 0
66)  3976   Main:        Semaphore delete status: 1
67)  3976   Main:        Main Finished
68)  3976   Counter:     Closing Semaphore: Id = 0x60005
```

4.6.5 その他のバリエーション

この例のその他のバリエーションも可能です。同時に実行できる Semaphore.Main は 1 つのみですが、他のウィンドウで実行される Producer または Consumer の数には制約はありません。以下のようにさまざまなシナリオでコンシューマとプロデューサの数を変化させようとすることをお勧めします。

- セマフォの“競合”を増やすために、3 つのコンシューマと 1 つのプロデューサの実行を試行します。運が良ければ、2 つ以上のコンシューマがセマフォのデクリメントを要求し、セマフォの値が十分に大きく、両方の要求の一部または全部を満たすことから両方が成功したことがログに示されます。
- これらのクラスを使用して、該当のセマフォを削除したときに他のプロセスで起こることを示すこともできます。これを行うには、Producer または Consumer の実行中に、Main クラスが実行されているウィンドウに切り替え、ENTER キーを押します。この処理を終了する過程で、Main クラスはセマフォを削除し、Producer または Consumer の OREF が有効でなくなります。次回使用しようとするとエラーが発生します。
- セマフォの名前をグローバル名のような名前に変更することによって、セマフォを別のインスタンス (ECP システムなど) にマップできます。