Lock Manager VMS

Overview

J.Y. Collot

Symposium DECUS

Juin 1991

YLYS
INFORMATIQUE

Table des Matières

1. Fonctionnalités
   a. Synchronisation
   b. Communication
2. Ressources et Locks
   a. Ressources
   b. Création et Conversion de Locks
   c. Deadlocks
3. Utilisation
   a. Synchronisation simple
   b. Blocking ASTs
   c. Value Blocks
4. System Services
5. Exemple

Fonctionnalités

Le Lock Manager permet de synchroniser l'accès à des ressources partagées par différents process.

Le Lock Manager permet aussi de communiquer entre process :

- des évènements
- des données (maximum 16 octets)

Le Lock Manager est un outil standard utilisé par de nombreux modules de VMS (XQP, RMS, files d'attente, etc...) et de nombreux produits (RDB,...)

Fonctionnalités

Les process peuvent s'exécuter sur des noeuds différents d'un même cluster.

	Synchro	Comm.	Cluster
Noms logiques		X	
Event Flags	X		
Mailboxes	X	X	
Sections Globales		X	
Lock Manager	X	X	X
Fichiers		X	X
DECnet	X	X	X

Ressources et Locks

Ressource

Une ressource est définie par l'utilisateur du Lock Manager par :

- un nom
- un mode d'accès (User, Super,...)
- une ressource "mère" (éventuellement)

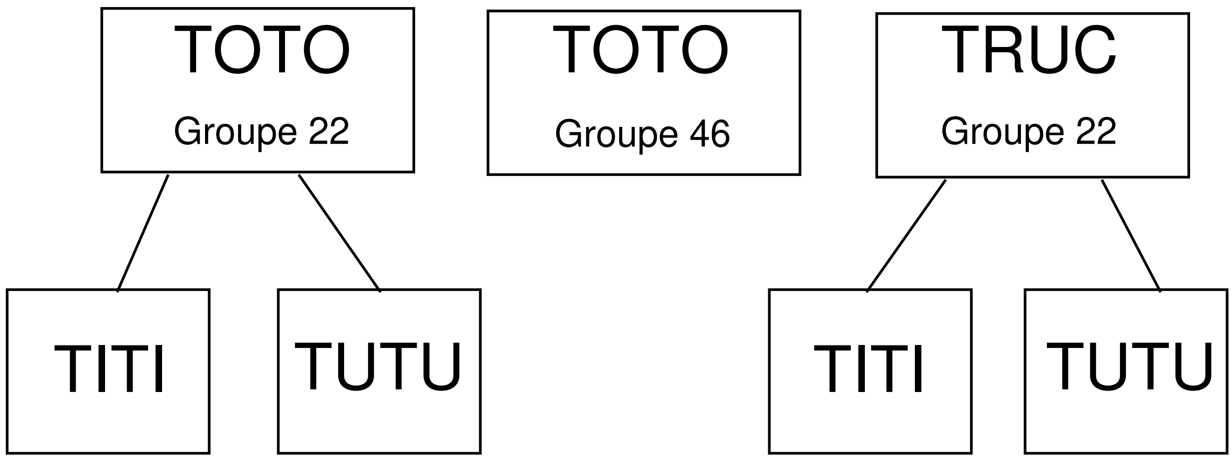
Une ressource peut être connue :

- par un groupe UIC
- par le système (privilège SYSLCK)

La ressource est créée par $ENQ lors du premier appel à ce Service pour une ressource inexistante et détruite par le $DEQ du dernier lock sur cette ressource.

Ressources et Locks

Exemples de ressources



Les différents process désirant se synchroniser doivent coopérer et donc être d'accord sur la ressource utilisée.

Il n'existe pas de "lien" entre la ressource et l'objet physique protégé (section, fichier...). L'association ressource <-> objet est uniquement applicative : elle n'est pas connue du système.

Ressources et Locks

La compatibilité entre les modes est donnée par le tableau suivant :

	NL	CR	CW	PR	PW	EX
NL	X	X	X	X	X	X
CR	X	X	X	X	X	
CW	X	X	X			
PR	X	X		X		
PW	X	X				
EX	X					

Il est possible de convertir un lock d'un mode X en mode Y.

L'opération de conversion est plus rapide que l'opération de création d'un nouveau lock.

Création

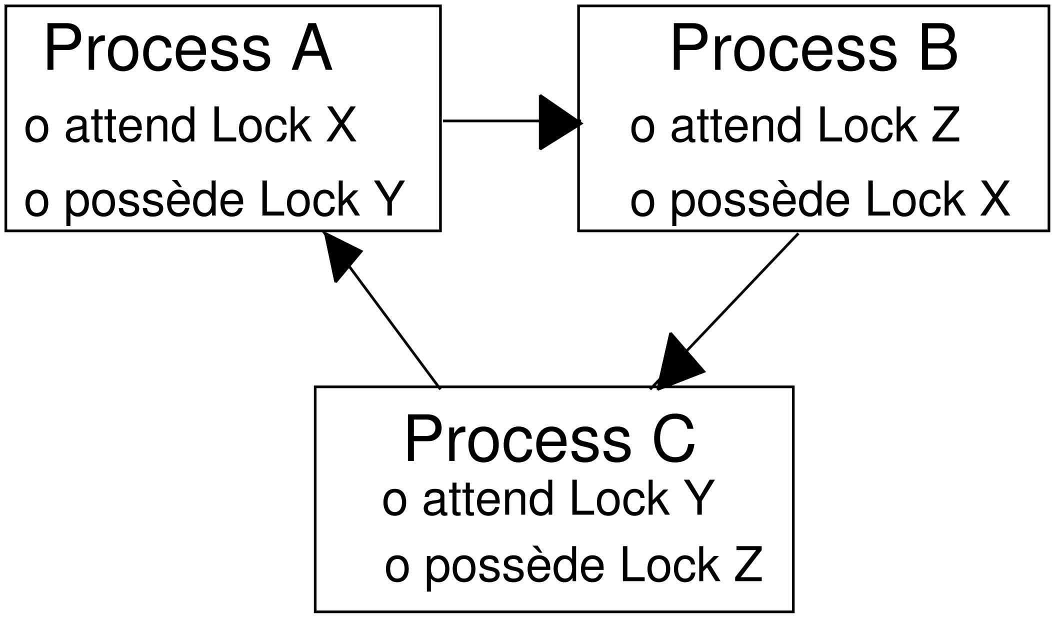
```
INCLUDE '($LCKDEF)'
INTEGER*2 LckBlk(2)
INTEGER Code, Sys$Enqw
Code = Sys$Enqw ( ,
1                   %val(LCK$K_NLMODE)
1                   LckBlk,
1                   ,
1                   'MACHIN',
1                   ,,,,,)
IF (Code) Code=LckBlk(1)
If (.NOT.Code) Call LIB$SIGNAL(%val(Code))
```

Conversions

```
          Code = Sys$Enqw ( ,
          1                   %val(LCK$K_EXMODE)
          1                   LckBlk,
          1                   %val(LCK$M_CONVERT),
          1                   ,
          1                   ,,,,,)
!
! Modifications des données partagées .....
!
          Code = Sys$Enqw ( ,
          1                   %val(LCK$K_NLMODE)
          1                   LckBlk,
          1                   %val(LCK$M_CONVERT),
          1                   ,
          1                   ,,,,,)
```

Deadlocks

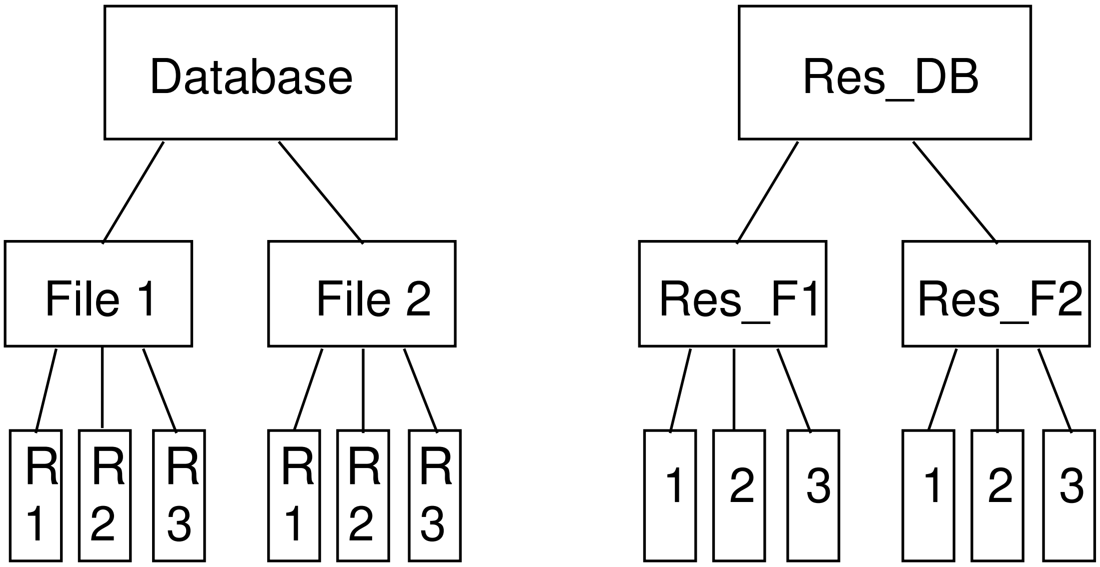
Un *Deadlock* est une situation dans laquelle plusieurs process s’attendent l’un l’autre.



Dans ce cas, le lock manager choisit une "victime" et renvoie le code d’erreur SS$_DEADLOCK

SubLocks

La notion de Sublock (et de Subressource) permet de calquer la structure des locks sur la structure des données partagées correspondantes.



On ne peut détruire un lock que lorsque tous ses sublocks sont détruits.

Synchronisation simple

Accès concurrent en lecture/écriture à une zone de données partagées.

Pour accès en lecture

1. Conversion mode NL -> mode PR
2. Lecture donnée
3. Conversion mode PR -> mode NL

Pour accès en écriture

1. Conversion mode NL -> mode PW
2. Modification donnée
3. Conversion mode PW -> mode NL

NB :

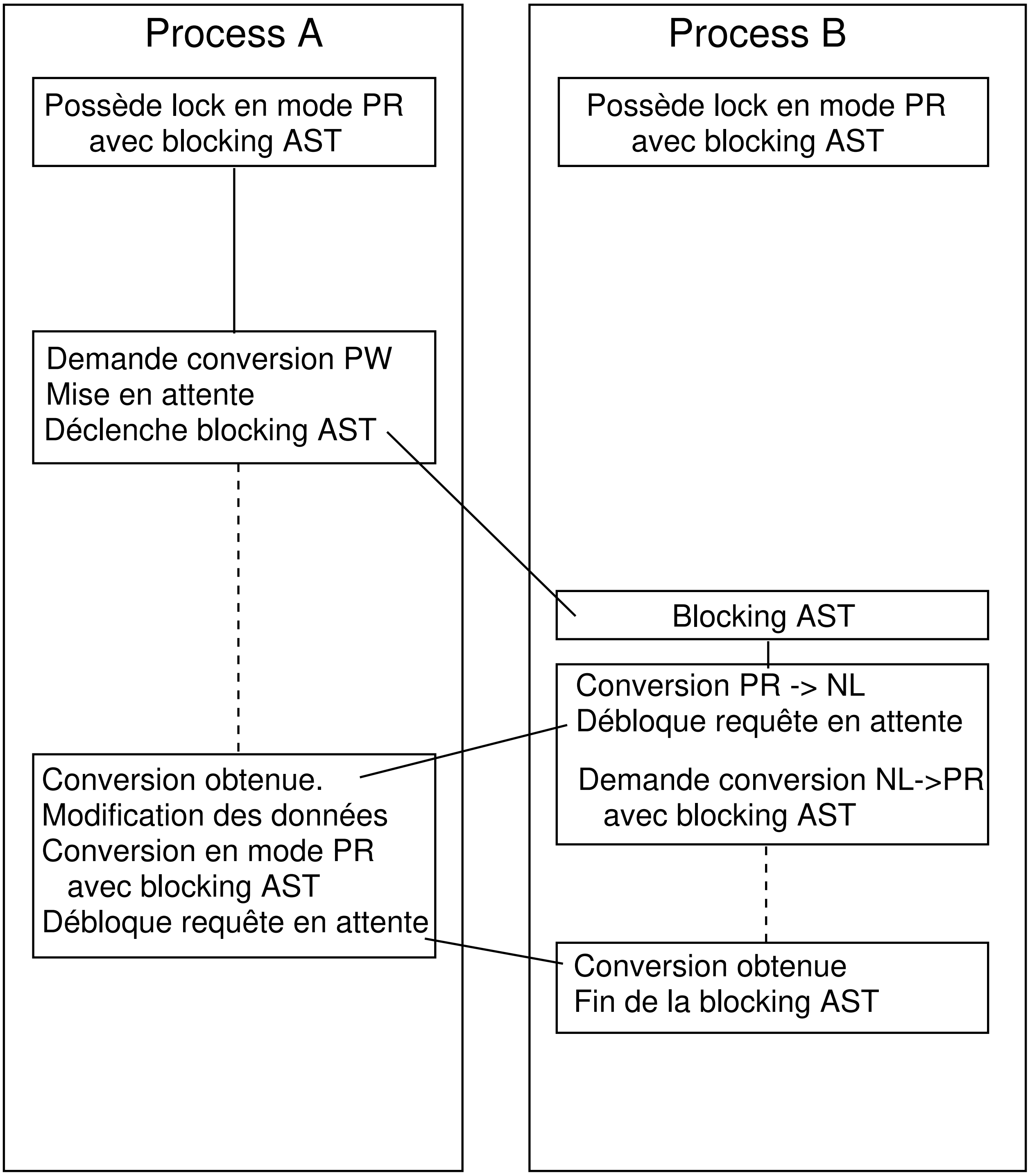
- Plusieurs modes PR sont compatibles
- Le mode PW est incompatible avec PW et avec PR

Dans l'exemple précédent, chaque accès en lecture demande 2 conversions de mode du lock correspondant.

Si l'on est dans un cas où les process font beaucoup d'accès en lecture et peu de modifications, cette méthode est lourde.

Il serait préférable de garder le lock en mode PR, et d'être prévenu lorsqu'un autre process désire le lock en mode PW. Ainsi, les conversions de mode ne seraient nécessaires que lors des (rares) modifications.

Le Lock Manager fournit cette fonctionnalité sous la forme de *Blocking AST*


Process A
Process B
Possède lock en mode PR avec blocking AST
Possède lock en mode PR avec blocking AST
Demande conversion PW
Mise en attente
Déclenche blocking AST
Blocking AST
Conversion PR -> NL
Débloque requête en attente
Demande conversion NL->PR avec blocking AST
Conversion obtenue.
Modification des données
Conversion en mode PR avec blocking AST
Débloque requête en attente
Conversion obtenue
Fin de la blocking AST

Blocking AST

Lorsque l’on prend un lock, il est possible de déclarer une routine (*blocking AST*) qui sera appelée si un autre process tente d’avoir un lock sur la même ressource, dans un mode incompatible avec celui dans lequel on tient ce lock.

Cette routine doit libérer le lock (afin de "laisser passer" la requête), puis se remettre dans l’état antérieur.

Les blocking ASTs permettent aussi de passer des évènements quelconques d’un process à un autre (cluster-wide).

Value Block

Le *Value Block* est une zone de 16 octets associée à la ressource, que l'on peut accéder en utilisant de flag LCK$M_VALBLK lors de l'appel à $ENQ.

Le programme appelant reçoit une copie à jour de cette zone à chaque création d'un nouveau lock ou conversion d'un mode X en mode Y.

Il peut modifier le contenu de cette zone en effectuant une conversion de mode EX ou PW en un autre mode.

Le Value Block permet de communiquer jusqu'à 16 octets (par lock) entre plusieurs process cluster-wide.

System Services

L'interface de programmation du Lock Manager se compose de trois system services :

- SYS$ENQW / SYS$ENQ
  Création ou conversion de mode, avec ou sans attente
- SYS$DEQ
  Destruction d'un ou de tous les locks d'un process
- SYS$GETLKIW / SYS$GETLKI
  Obtention d'informations sur un lock ou sur une ressource avec ou sans attente.

Lock Manager VMS

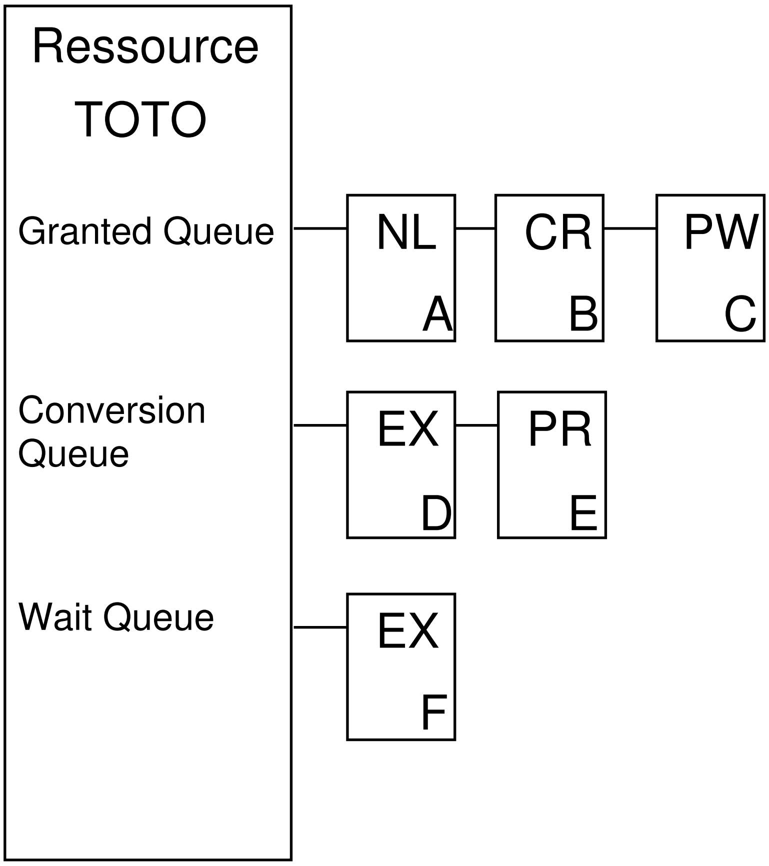
```
SYS$ENQ(W)  ( Event_Flag,
              Mode,
              Lock_Block,
              Flags,
              [Ressource],
              [Lock_Parent],
              [Ast_Completion],
              [Ast_Param],
              [Blocking_Ast],
              [Access_Mode])
```

Format du Lock Block

Code Retour VMS
Lock Ident
Value Block (16 octets) (seulement si flag LCK$M_VALBLK)

Pour chaque ressource, le lock manager conserve 3 queues :

- Une queue des locks "Granted",
- Une queue des locks en attente de conversion, dont le nouveau mode est incompatible avec les locks "granted",
- Une queue des nouveaux locks ne pouvant pas être "granted" actuellement.



Lors d’une conversion, celle-ci est immédiatement "granted" si les autres locks restant "granted" sont compatibles avec le nouveau mode.

Par exemple, une conversion NL->CR est immédiatement "granted" si les autres locks "granted" sont tous compatibles avec CR, même s’il existe des locks incompatibles en attente dans la queue "conversion" ou "wait". Le flag LCK$M_QUECVT force le passage par la queue "conversion"

Un nouveau lock n’est "granted" que si

- son mode est compatible avec les autres locks "granted"
- il n’y a pas d’autres locks en attente dans les queues "conversion" ou "wait"

Ainsi, un nouveau lock, même en mode NL peut être mis en attente, s’il existe par exemple un lock CR "granted", et un lock CR->EX dans la queue "conversion. Le flag LCK$M_EXPEDITE permet d’obtenir immédiatement le lock NL dans tous les cas.

Principaux Flags de SYS$ENQ(W)

- LCK$M_NOQUEUE
  Si le lock n'est pas obtenu immédiatement, il n'est pas mis en queue.

- LCK$M_SYSTEM
  Le lock est "system-wide" (non limité à un groupe).

- LCK$M_VALBLK
  Le lock block contient une zone "value block".

- LCK$M_CONVERT
  La requête est une conversion de mode.

- LCK$M_QUECVT
  Force le passage par la queue de conversion, même si le nouveau mode est compatible avec les locks déjà "granted"

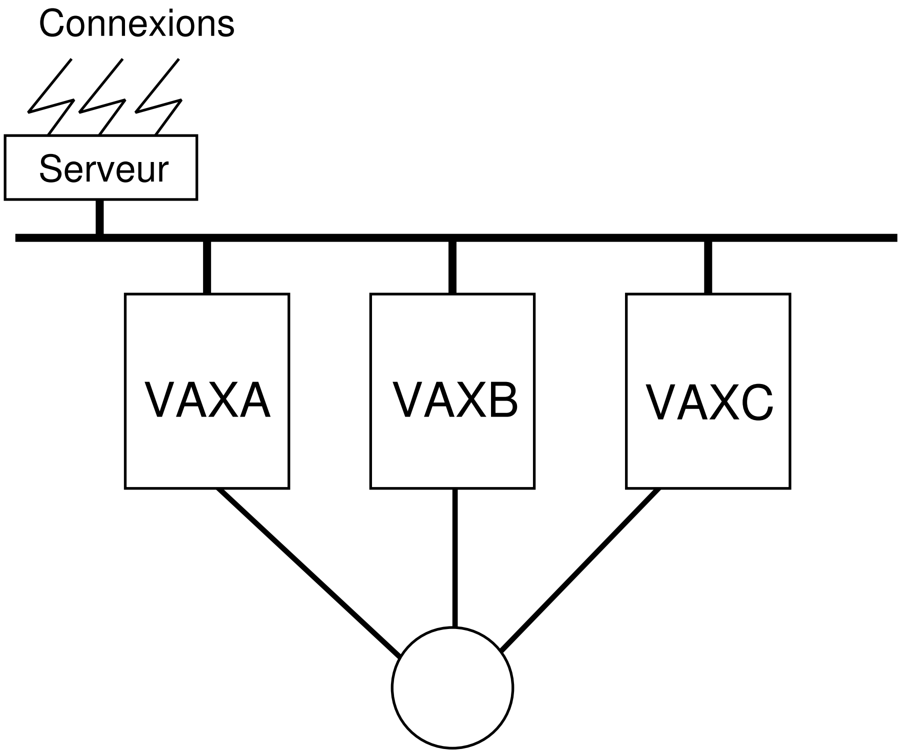
- LCK$M_EXPEDITE
  Lors de la création d'un nouveau lock en mode NL, obtient le lock immédiatement, même s'il existe d'autres locks dans les queues "wait" ou "conversion"

```
SYS$DEQ  ( [Lock_Id],
           [Value_Block],
           [Access_Mode],
           [Flags],)
```

Principaux flags de SYS$DEQ

- LCK$M_DEQALL
  Détruit tous les locks actuellement tenus par ce process
- LCK$M_CANCEL
  Détruit une requête de conversion ou de création de lock actuellement en attente (non "granted")

Exemple



Un certain nombre de lignes asynchrones d'un ou plusieurs serveurs de terminaux sont reliés à une machine X avec laquelle on peut faire des transferts de fichiers.

L'application cluster doit :

- choisir une des lignes libres avant de faire le transfert,
- fournir un utilitaire permettant de
  - -- connaître le username des utilisateurs des lignes occupées,
  - -- tuer un transfert en cours

Exemple (suite)

A chaque ligne disponible correspond une ressource dont le nom inclut le nom physique du port application défini (par exemple RES_LTAxx).

Lorsque l'on désire effectuer un transfert, on tente d'abord de créer un lock en mode EX sur chacune des ressources connues (RES_LTA1, RES_LTA2, ...). Pour ce lock, on utilise les flags VALBLK, SYSTEM et NO-QUEUE.

Lorsque l'on a obtenu un lock, on modifie le "value block" en y copiant son "username", puis en faisant une conversion EX->EX.

Ce lock est alors associé à une "blocking AST". Le code de cette routine contient les opérations d'abort de transfert, puis l'appel à SYS$EXIT.

Pour connaître l'utilisation des lignes, il suffit d'un programme qui crée un lock en mode NL sur chacune des ressources. Le "value block" contient soit 0 (ligne libre) soit le username de l'utilisateur courant.

Pour tuer un transfert en cours sur la ligne XXX, il suffit de demander la création d'un lock de mode incompatible avec EX sur la ressource RES_XXX. Cette requête active la blocking AST dans le process de transfert.