

## TP 5 : étude du protocole Xen vchan

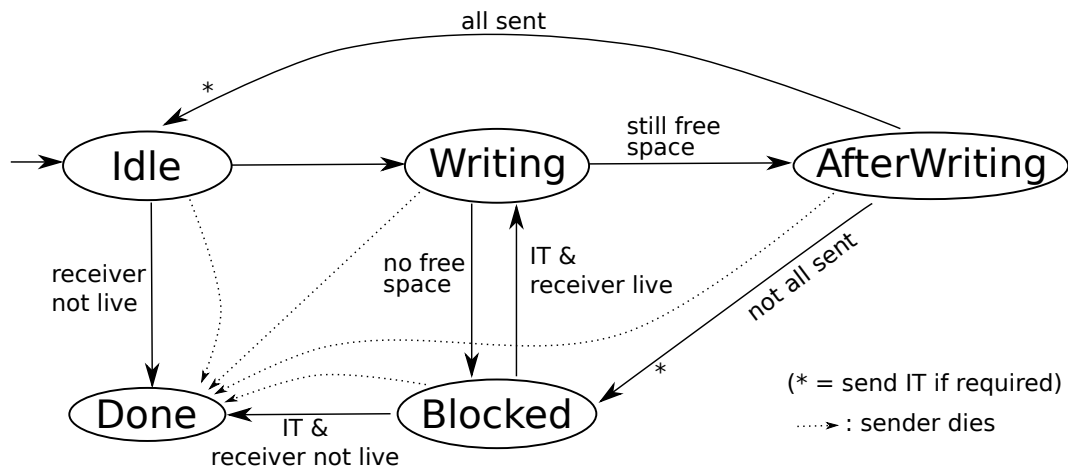
*vchan* est un protocole de communication en mémoire entre machines virtuelles utilisé par l'hyperviseur Xen. L'objectif est d'en étudier une version à peine simplifiée.

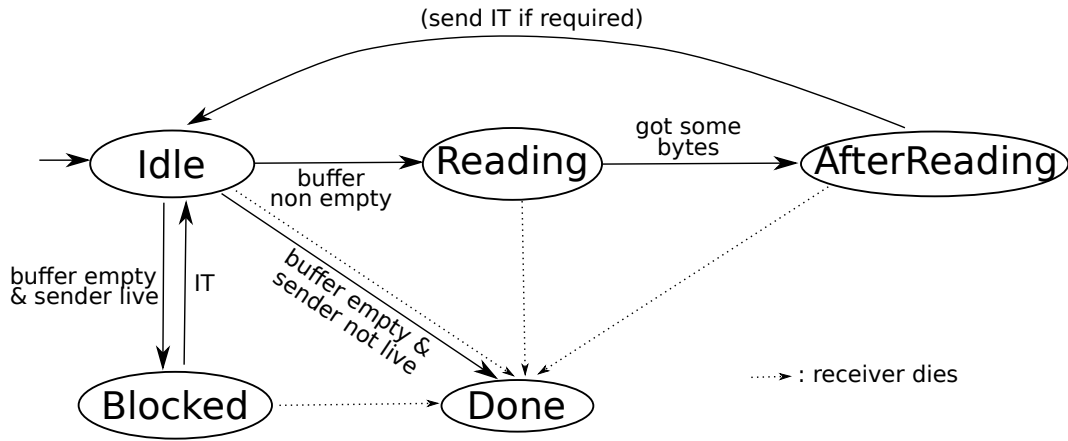
Le système est composé d'un émetteur et d'un récepteur. L'émetteur dépose des octets dans un buffer, et le récepteur va lire dans ce buffer. Quand le buffer est plein, l'émetteur se bloque ; quand le buffer est vide, le récepteur se bloque. Un mécanisme d'interruption permet à l'émetteur de signaler qu'il a déposé des données dans le buffer, et au récepteur de signaler qu'il a libéré de la place dans le buffer.

Plusieurs difficultés sont à prendre en compte :

- L'émetteur envoie successivement des messages composés d'octets ; la taille d'un message n'est pas fixée.
- Il n'y a pas de taille fixe de lecture et/ou écriture : le message à envoyer est découpé en un nombre arbitraire d'ajouts successifs dans le buffer (selon la place disponible dans celui-ci) ; le récepteur retire un nombre arbitraire d'octets du buffer, pas nécessairement tous en une lecture. On dispose néanmoins d'une borne supérieure sur le nombre d'octets écrits (**MaxWriteLen**) et lus (**MaxReadLen**) en une unique action.
- À tout moment, le récepteur ou l'émetteur peuvent mourir. Une interruption est alors envoyé à l'autre pair. Celui-ci devra à son tour mourir (en ayant d'abord vidé le buffer pour le récepteur).
- Des interruptions parasites (*spurious IT*) peuvent survenir à tout moment.

Les automates complets des deux pairs sont présentés ci-dessous :

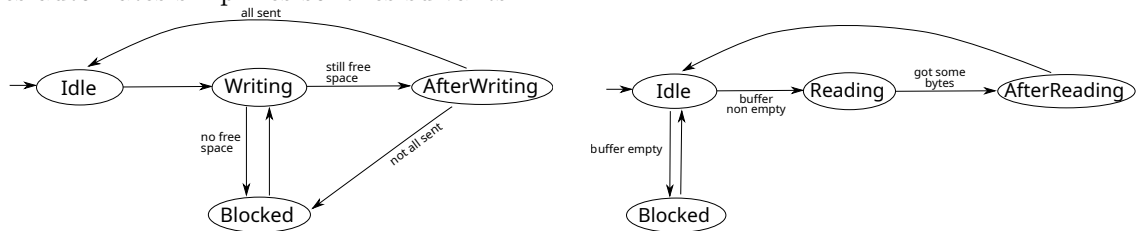




## À faire

- Étudier les propriétés attendues :
  - *Integrity* : ce qu'on reçoit est un préfixe de ce qui a été envoyé (*Got* est un préfixe de *Sent*) ;
  - *Availability* : quelque soit le nombre d'octets envoyés, on finira par en avoir reçu au moins autant ;
  - *NoLoss* : tout ce qui est envoyé finira par être reçu.
- Faire un premier modèle :
  - sans interruptions : les variables *NotifyRead* et *NotifyWrite* ne sont jamais positionnées et donc *SenderIT* et *ReceiverIT* restent toujours à FALSE. On quitte sans condition les états *Blocked*.
  - sans mort d'un pair : *SenderLive* et *ReceiverLive* restent toujours vraies et les transitions vers *Done* sont pour le moment ignorées.

Les automates simplifiés sont les suivants :

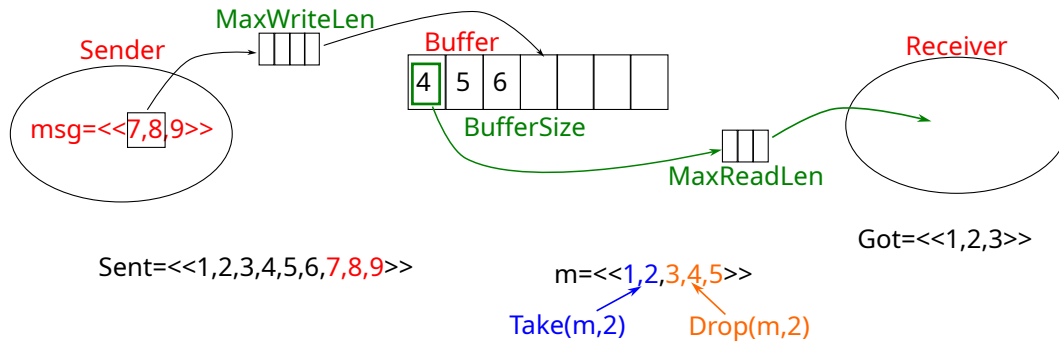


- Vérifier si les propriétés attendues sont vérifiées.
- Introduire l'attente sur interruption : positionnement de *NotifyRead* et *NotifyWrite* et utilisation de *SenderIT* et *ReceiverIT*.
- Introduire la terminaison des pairs : transitions *SenderClose* et *ReceiverClose*, transitions vers *Done*, ajout de *ReceiverLive*/*SenderLive* dans les gardes des autres transitions.
- Modifier les propriétés *Availability* et *NoLoss* pour prendre en compte la mort d'un pair.
- Énoncer que si l'un quelconque des pairs meurt, alors les deux pairs finiront dans l'état *Done* et vérifier cette propriété.
- Introduire les interruptions parasites et vérifier le modèle final.

## Remarques sur le modèle TLA<sup>+</sup>

### Les variables et les constantes

- **Sent** et **Got** sont des variables d'histoire qui servent exclusivement à énoncer les propriétés attendues par le modèle.
- **msg** est ce qui reste à envoyer, **Buffer** est ce qui en transit.
- La constante **MaxWriteLen** indique le nombre maximal d'octets qu'on peut transférer du **msg** au **Buffer**, et **MaxReadLen** est le nombre maximal d'octets que le récepteur peut prendre dans le buffer.



**Interruptions de communication :** les interruptions sont modélisées par deux variables booléennes *SenderIT* et *ReceiverIT* qui indiquent respectivement que l'émetteur / le récepteur a une interruption en attente. Le pair concerné remet sa variable à faux quand il a traité l'interruption. Les variables *NotifyWrite* / *NotifyRead* permettent au récepteur / à l'émetteur de demander l'envoi d'une interruption quand l'opération correspondante a lieu.

**Interruptions de terminaison :** deux booléens *SenderLive* et *ReceiverLive* indiquent si l'émetteur (resp. le récepteur) sont vivants. Chaque pair peut mourir brutalement à tout moment (action *SenderClose* ou *ReceiverClose*). L'interruption *SenderIT* / *ReceiverIT* doit alors être positionnée pour informer l'autre participant.

**Model-checking :** tel qu'énoncé, le problème est non borné. On vérifiera avec les valeurs suivantes (ces modèles sont fournis) :

1. Version minimale :
  - $\text{MaxReadLen} = 2$
  - $\text{MaxWriteLen} = 1$  ou  $2$
  - $\text{BufferSize} = 1$  ou  $2$
  - $\text{Byte} = 1..5$
  - Modèle  $\triangleright$  *Advanced Options*  $\triangleright$  *State Constraint* :  $\text{Len}(\text{Sent}) < 4$
2. Version avancée :
  - $\text{MaxReadLen} = 3$
  - $\text{MaxWriteLen} = 3$
  - $\text{BufferSize} = 4$
  - $\text{Byte} = 1..9$
  - *State Constraint* :  $\text{Len}(\text{Sent}) < 6$

Note : `vchan_model1.tla` et `vchan_model2.tla` servent pour VSCode ou en ligne de commande. Ils ne servent à rien avec la TLA<sup>+</sup> toolbox.

Ce TP est inspiré de <http://roscidus.com/blog/blog/2019/01/01/using-tla-plus-to-understand-xen->  
et <https://github.com/talex5/spec-vchan/>