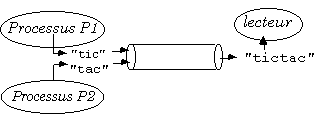
*Chapitre 3 : La communication interprocessus*

* Partage de variables (modèles : producteur/ consommateur, lecteurs/ rédacteurs)
* Boites aux lettres
* Echange de messages (modèle du client/ serveur)

1. **Introduction**

Les processus ont souvent besoin de communiquer avec d'autres processus. Par exemple, dans un pipeline shell, la sortie du premier processus doit être transmise au deuxième processus, et ainsi de suite. Il est donc nécessaire que les processus communiquent entre eux, de préférence d'une manière bien structurée et sans utiliser d'interruptions.



*Figure 9 Communication des processus*

Cette communication entre processus sous le contrôle du système d'exploitation est appelée *communication interprocessus ou simplement IPC (Inter processus communication).*

Dans certains systèmes d'exploitation, les processus qui travaillent ensemble partagent fréquemment une zone de stockage commune que chacun peut lire et écrire.

Pour voir comment l'IPC fonctionne en pratique, considérons un exemple simple, mais courant, un *spooler d'impression*. Lorsqu'un processus veut imprimer un fichier, il entre le nom du fichier dans un *répertoire* spécial du *spooler*. Un autre processus, le *démon d'impression*, vérifie périodiquement s'il y a des fichiers à imprimer, et s'il y en a, il les envoie à l'imprimante et supprime leurs noms du répertoire.

Des processus actifs simultanément sont dits exécutés en parallèle. Ils peuvent opérer de façon tout à fait indépendante, mais une interaction entre processus parallèles est souvent nécessaire. L’origine de cette nécessité d’interaction peut être :

* La coopération à un but commun, par exemple
  + Le calcul parallèle sur une machine à processeurs multiples,
  + La coopération entre processus se trouvant sur des machines différentes dans le contexte d’applications distribuées,
  + L’organisation d’un programme en tâches parallèles, notamment dans le cas d’un système de contrôle ;
* Le partage de ressources, par exemple un système de fichiers sur disque.

1. **Communication inter-processus (ipc)**

Un processus peut être de deux types :

* Processus indépendant
* Processus de coopération

Un processus indépendant n'est pas affecté par l'exécution d'autres processus, tandis qu'un processus coopérant peut être affecté par d'autres processus en cours d'exécution. Bien que l'on puisse penser que ces processus, qui s'exécutent indépendamment, s'exécuteront très efficacement, en réalité, il existe de nombreuses situations où la nature coopérative peut être utilisée pour augmenter la vitesse de calcul, la commodité et la modularité.

La communication inter-processus (IPC) est un mécanisme qui permet aux processus de communiquer entre eux et de synchroniser leurs actions. La communication entre ces processus peut être vue comme une méthode de coopération entre eux. Les processus peuvent communiquer entre eux via :

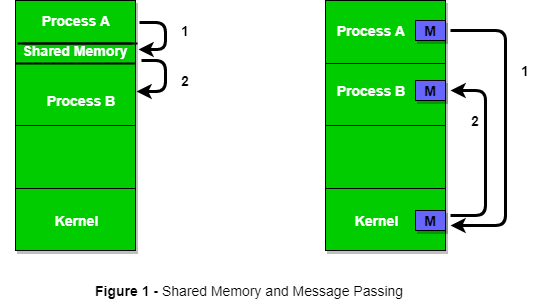
1. La mémoire partagée
2. La transmission de messages

La figure ci-dessous montre une structure de base de communication entre processus via *la méthode de la mémoire partagée* et via *la méthode de transmission de messages*. Un système d'exploitation peut implémenter les deux méthodes de communication.

Tout d'abord, nous discuterons des méthodes de communication de la mémoire partagée, puis de la transmission des messages. La communication entre processus utilisant la mémoire partagée nécessite que les processus partagent une variable, et cela dépend entièrement de la façon dont le programmeur l'implémentera. Une manière de communiquer utilisant la mémoire partagée peut être imaginée comme ceci : supposons que *process1* et *process2* s'exécutent simultanément, et qu'ils partagent certaines ressources ou utilisent des informations d'un autre processus.

*Process1* génère des informations sur certains calculs ou ressources utilisés et les conserve sous forme d'enregistrement dans la mémoire partagée. Lorsque *process2* a besoin d'utiliser les informations partagées, il vérifiera l'enregistrement stocké dans la mémoire partagée et prendra note des informations générées par *process1* et agira en conséquence.

Dans ce qui suit, nous examinons un exemple de communication entre processus utilisant la méthode de la mémoire partagée.



*Figure 10 Mémoire partagée et passage de messages*

* 1. **Méthode de mémoire partagée**

La communication interprocessus par le biais de la mémoire partagée est un concept dans lequel deux ou plusieurs processus peuvent accéder à la mémoire commune. Et la communication se fait via cette mémoire partagée, où les modifications apportées par un processus peuvent être vues par un autre processus.

Le problème avec les pipes, fifo et les files d'attente de messages - est que pour que deux processus échangent des informations. L'information doit passer par le noyau.

Le serveur lit le fichier d'entrée.

Le serveur écrit ces données dans un message en utilisant soit un pipe, un fifo ou une file de messages.

Le client lit les données depuis le canal IPC, ce qui nécessite de copier les données depuis le tampon IPC du noyau vers le tampon du client.

Enfin, les données sont copiées à partir du tampon du client.

Au total, quatre copies de données sont nécessaires (2 en lecture et 2 en écriture). La mémoire partagée fournit donc un moyen de permettre à deux ou plusieurs processus de partager un segment de mémoire. Avec la mémoire partagée, les données ne sont copiées que deux fois - du fichier d'entrée vers la mémoire partagée et de la mémoire partagée vers le fichier de sortie.

| ***ftok()*** *: est utilisé pour générer une clé unique.*  ***shmget()*** *: int shmget(key\_t,size\_tsize,intshmflg) ; en cas de réussite, shmget() retourne un identifiant pour le segment de mémoire partagée.*  ***shmat()*** *: Avant de pouvoir utiliser un segment de mémoire partagée, vous devez vous attacher*  *à l'aide de shmat(). void \*shmat(int shmid ,void \*shmaddr ,int shmflg) ;*  *shmid est l'identifiant de la mémoire partagée. shmaddr spécifie l'adresse spécifique à utiliser mais nous devrions la mettre à zéro et le système d'exploitation choisira automatiquement le segment de mémoire partagée.*  *mais nous devrions le mettre à zéro et le système d'exploitation choisira automatiquement l'adresse.*  ***shmdt()*** *: Lorsque vous en avez fini avec le segment de mémoire partagée, votre programme devrait*  *se détacher de celui-ci en utilisant shmdt(). int shmdt(void \*shmaddr) ;*  ***shmctl()*** *: quand vous vous détachez de la mémoire partagée, elle n'est pas détruite. Donc, pour détruire*  *shmctl() est utilisé. shmctl(int shmid,IPC\_RMID,NULL) ;* |
| --- |

**Ex : Problème Producteur-Consommateur**

Il existe deux processus : Producteur et Consommateur. Le producteur produit certains articles et le consommateur consomme cet article. Les deux processus partagent un espace commun ou un emplacement mémoire appelé tampon où l'article produit par le Producteur est stocké et à partir duquel le Consommateur consomme l'article si nécessaire.

Il existe deux versions de ce problème :

* La première est connue sous le nom de problème de tampon illimité dans lequel le producteur peut continuer à produire des éléments et il n'y a pas de limite à la taille du tampon,
* La seconde est connue sous le nom de problème de tampon borné dans que le Producteur peut produire jusqu'à un certain nombre d'articles avant de commencer à attendre que le Consommateur le consomme.

Nous aborderons le problème du tampon borné. Premièrement, le producteur et le consommateur partageront une mémoire commune, puis le producteur commencera à produire des articles. Si l'article produit total est égal à la taille du tampon, le producteur attendra de le faire consommer par le consommateur. De même, le consommateur vérifiera au préalable la disponibilité de l'article. Si aucun article n'est disponible, le Consommateur attendra que le Producteur le produise. S'il y a des articles disponibles, le consommateur les consommera. Le pseudo-code à démontrer est fourni ci-dessous :

* **Données partagées entre les deux processus en C**

**Code de processus du producteur**

| #define buff\_max 25  #define mod %      struct item{      // membre différent des données produites ou des données consommées      }       // Un tableau est nécessaire pour contenir les éléments. C'est le  lieu partagé auquel les deux processus auront accès.      // item shared\_buff [ buff\_max ];        // Deux variables qui garderont la trace des indices des articles  produits par le producteur et le consommateur, L'indice libre  pointe vers le prochain indice libre. L'indice complet pointe  vers le premier indice complet      int free\_index = 0;      int full\_index = 0; |
| --- |

**Code de processus du producteur en C**

| item nextProduced;      while(1){  // vérifier s'il n'y a pas d'espace pour la production.          // Si oui, continuez à attendre.          while((free\_index+1) mod buff\_max == full\_index);           shared\_buff[free\_index] = nextProduced;           free\_index = (free\_index + 1) mod buff\_max;      } |
| --- |

**Code de processus consommateur en C**

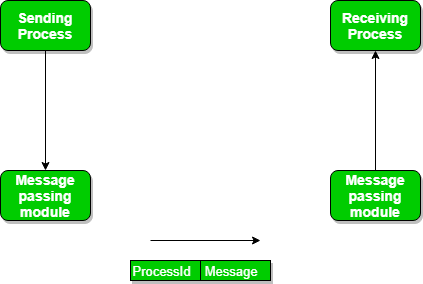
| item nextConsumed;      while(1){          // vérifier s'il y a un article disponible pour la consommation.  // Si ce n'est pas le cas, continuez à attendre.  while((free\_index == full\_index);           nextConsumed = shared\_buff[full\_index];           full\_index = (full\_index + 1) mod buff\_max;      } |
| --- |

Dans le code ci-dessus, le Producteur recommencera à produire lorsque le mod buff max (free\_index+1) sera gratuit, car s'il n'est pas gratuit, cela implique qu'il y a encore des objets qui peuvent être consommés par le Consommateur, donc il n'y a pas besoin produire plus. De même, si l'index libre et l'index complet pointent vers le même index, cela implique qu'il n'y a pas d'éléments à consommer.

* 1. **Méthode de transmission de messagerie**

Maintenant, nous allons commencer notre discussion sur la communication entre les processus via la transmission de messages. Dans cette méthode, les processus communiquent entre eux sans utiliser aucun type de mémoire partagée. Si deux processus p1 et p2 veulent communiquer entre eux, ils procèdent comme suit :

* Établir un lien de communication (si un lien existe déjà, pas besoin de l'établir à nouveau.)
* Commencez à échanger des messages en utilisant des primitives de base. Nous avons besoin d'au moins deux primitives :
  + **envoyer** (message, destination) ou **envoyer** (message)
  + **recevoir** (message, hôte) ou **recevoir** (message)



*Figure 11 Méthode de transmission de messagerie*

La figure 11, ci-dessous, illustre le mécanisme de communication par échange de messages.

* La taille du message peut être de taille fixe ou de taille variable. S'il est de taille fixe, c'est facile pour un concepteur d'OS, mais compliqué pour un programmeur, et s'il est de taille variable, alors c'est facile pour un programmeur, mais compliqué pour le concepteur d'OS. Un message standard peut avoir deux parties : un **en-tête et un corps.**
* La **partie en-tête** est utilisée pour stocker le type de message, l'identifiant de destination, l'identifiant de source, la longueur du message et les informations de contrôle. Les informations de contrôle contiennent des informations telles que ce qu'il faut faire en cas de manque d'espace tampon, le numéro de séquence, la priorité. Généralement, le message est envoyé en utilisant le style FIFO.
  1. **Message passant par le lien de communication**

**Lien de communication direct et indirect**

Maintenant, nous allons commencer notre discussion sur les méthodes de mise en œuvre des liens de communication. Lors de la mise en œuvre du lien, certaines questions doivent être prises en compte, telles que :

* Comment les liens sont-ils établis ?
* Un lien peut-il être associé à plus de deux processus ?
* Combien de liens peut-il y avoir entre chaque paire de processus communicants ?
* Qu'elle est la capacité d'un lien ? La taille d'un message que le lien peut accepter est-elle fixe ou variable ?
* Un lien est-il unidirectionnel ou bidirectionnel ?

Un lien a une certaine potentialité qui détermine le nombre de messages qui peuvent y résider temporairement, pour lesquels chaque lien est associé à une file d'attente qui peut être de possibilité nulle, de possibilité limitée ou de possibilité illimitée. Au maximum de sa possibilité nulle, l'expéditeur attend que le destinataire informe l'expéditeur qu'il a reçu le message. Dans les cas de disposition non nulle, un processus ne sait pas si un message a été reçu ou non après l'opération d'envoi. Pour cela, l'expéditeur doit communiquer explicitement avec le destinataire. La mise en œuvre du lien dépend de la situation, il peut s'agir soit d'un lien de communication direct, soit d'un lien de communication indirect.

* **Liens de communication directe** sont mis en œuvre lorsque les processus utilisent un identifiant de processus spécifique pour la communication, mais il est difficile d'identifier l'expéditeur à l'avance.

**Exemple :** le serveur d'impression.

* **La communication indirecte** se fait via une boîte aux lettres partagée (port), qui consiste en une file d'attente de messages. L'expéditeur garde le message dans la boîte aux lettres et le destinataire le récupère.
  1. **Message passant par l'échange des messages.**

**Transmission de messages synchrone et asynchrone :**

Un processus bloqué est un processus qui attend un événement, tel qu'une ressource devenant disponible ou l'achèvement d'une opération d'E/S.

L'IPC est possible entre les processus sur le même ordinateur ainsi que sur les processus exécutés sur différents ordinateurs, c'est-à-dire dans un système en réseau/distribué.

Dans les deux cas, le processus peut ou non être bloqué lors de l'envoi d'un message ou de la tentative de réception d'un message, de sorte que la transmission du message peut être bloquante ou non bloquante. Le blocage est considéré comme synchrone et le blocage de l'envoi signifie que l'expéditeur sera bloqué jusqu'à ce que le message soit reçu par le destinataire. De même, le blocage de la réception bloque le récepteur jusqu'à ce qu'un message soit disponible. Non bloquant est considéré l'envoi asynchrone et non bloquant permet à l'expéditeur d'envoyer le message et de continuer.

Aussi, la réception non bloquante permet au destinataire de recevoir un message valide ou nul. Après une analyse minutieuse, nous pouvons conclure que pour un expéditeur, il est plus naturel d'être non bloquant après le passage du message, car il peut être nécessaire d'envoyer le message à différents processus. Cependant, l'expéditeur attend un accusé de réception de la part du destinataire en cas d'échec de l'envoi. De même, il est plus naturel pour un récepteur de bloquer après avoir émis la réception parce que les informations du message reçu peuvent être utilisées pour une exécution ultérieure. Dans le même temps, si l'envoi du message continu d'échouer, le destinataire devra attendre indéfiniment. C'est pourquoi nous considérons également l'autre possibilité de transmission de messages.

Il existe essentiellement trois combinaisons préférées :

* Blocage de l'envoi et blocage de la réception
* Envoi non bloquant et réception non bloquante
* Envoi non bloquant et réception bloquante (principalement utilisé)

Dans le passage de message direct, le processus qui veut communiquer doit nommer explicitement le destinataire ou l'expéditeur de la communication.

Par exemple, envoyer (p1, message) signifie envoyer le message à p1.

De même, recevoir (p2, message) signifie recevoir le message de p2.

Dans cette méthode de communication, la liaison de communication est établie automatiquement, qui peut être unidirectionnelle ou bidirectionnelle, mais une liaison peut être utilisée entre une paire d'expéditeur et de destinataire, et une paire d'expéditeur et de destinataire ne doit pas posséder plus d'une paire de liens. La symétrie et l'asymétrie entre l'envoi et la réception peuvent également être mises en œuvre, c'est-à-dire que les deux processus se nommeront pour l'envoi et la réception des messages, ou que seul l'expéditeur nommera le destinataire pour l'envoi du message et il n'est pas nécessaire que le destinataire nomme l'expéditeur pour réception du message.

Le problème avec cette méthode de communication est que si le nom d'un processus change, cette méthode ne fonctionnera pas.

Dans le passage de message indirect, les processus utilisent des boîtes aux lettres (aussi appelées ports) pour envoyer et recevoir des messages. Chaque boîte aux lettres à un identifiant unique et les processus ne peuvent communiquer que s'ils partagent une boîte aux lettres. Le lien établi uniquement si les processus partagent une boîte aux lettres commune et qu'un seul lien peut être associé à plusieurs processus. Chaque couple de processus peut partager plusieurs liens de communication et ces liens peuvent être unidirectionnels ou bidirectionnels.

Supposons que deux processus souhaitent communiquer via la transmission indirecte de messages, les opérations requises sont : créer une boîte aux lettres, utiliser cette boîte aux lettres pour envoyer et recevoir des messages, puis détruire la boîte aux lettres. Les primitives standard utilisées sont : send(A, message) qui signifie envoyer le message à la boîte aux lettres A.

La primitive de réception du message fonctionne également de la même manière, par exemple Receive (A, message).

Il y a un problème avec cette implémentation de boîte aux lettres. Supposons qu'il y ait plus de deux processus partageant la même boîte aux lettres et supposons que le processus p1 envoie un message à la boîte aux lettres, quel processus sera le destinataire ? Cela peut être résolu soit en imposant que seuls deux processus puissent partager une seule boîte aux lettres, soit en imposant qu'un seul processus, soit autorisé à exécuter la réception à un moment donné ou en sélectionnant n'importe quel processus au hasard et en informant l'expéditeur du destinataire.

Une boîte aux lettres peut être rendue privée à une seule paire expéditeur/destinataire et peut ainsi être partagée entre plusieurs paires expéditeur/destinataire. Le Port est une implémentation d'une telle boîte aux lettres qui peut avoir plusieurs expéditeurs et un seul destinataire. Il est utilisé dans les applications client/serveur (dans ce cas, le serveur est le récepteur). Le port appartient au processus récepteur et est créé par le système d'exploitation à la demande du processus récepteur et peut être détruit à la demande du même processeur récepteur lorsque le récepteur se termine lui-même, se faire en sorte qu'un seul processus soit autorisé à exécuter la réception peut être fait en utilisant le concept d'exclusion mutuelle. La boîte aux lettres *mutex* est créée et partagée par n processus. L'expéditeur est non bloquant et envoie le message. Le premier processus qui exécute la réception entrera dans la section critique et tous les autres processus seront bloquants et attendront.

Maintenant, discutons du problème Producteur-Consommateur en utilisant le concept de transmission de message.

Le producteur place des articles (à l'intérieur des messages) dans la boîte aux lettres et le consommateur peut consommer un article lorsqu'au moins un message est présent dans la boîte aux lettres. Le code est donné ci-dessous :

**Code Producteur en C**

| void Producer(void){  int item;  Message m;   while(1){      receive(Consumer, &m);      item = produce();      build\_message(&m , item ) ;      send(Consumer, &m);          }      } |
| --- |

**Code de la consommation** **en C**

| void Consumer(void){          int item;          Message m;          while(1){               receive(Producer, &m);              item = extracted\_item();              send(Producer, &m);              consume\_item(item);          }      } |
| --- |

* 1. **Exemples de systèmes IPC**
* Posix : utilise la méthode de la mémoire partagée.
* Mach : utilise le passage de messages
* Windows XP : utilise la transmission de messages à l'aide d'appels procéduraux locaux
  1. **Communication en architecture client/serveur :**

La communication client/serveur implique deux composants, à savoir un client et un serveur. Il s'agit généralement de plusieurs clients en communication avec un seul serveur. Les clients envoient des demandes au serveur et le serveur répond aux demandes des clients.

Il existe trois méthodes principales de communication client/serveur. Elles sont présentées comme suit .

Il existe différents mécanismes :

* + - **Pipe**

Il s'agit de méthodes de communication interprocessus qui contiennent deux points d'extrémité. Les données sont introduites à une extrémité du tuyau par un processus et consommées à l'autre extrémité par l'autre processus.

Les deux différents types de tuyaux sont les tuyaux ordinaires et les tuyaux nommés. Les tuyaux ordinaires ne permettent qu'une communication unidirectionnelle. Pour une communication bidirectionnelle, deux tuyaux sont nécessaires. Les pipes ordinaires ont une relation parent-enfant entre les processus, car les pipes ne peuvent être accédées que par les processus qui les ont créés ou hérités.

Les tubes nommés sont plus puissants que les tubes ordinaires et permettent une communication bidirectionnelle. Ces pipes existent même après la fin des processus qui les utilisent. Ils doivent être explicitement supprimés lorsqu'ils ne sont plus nécessaires.

Un diagramme illustrant les pipes est donné ci-dessous.

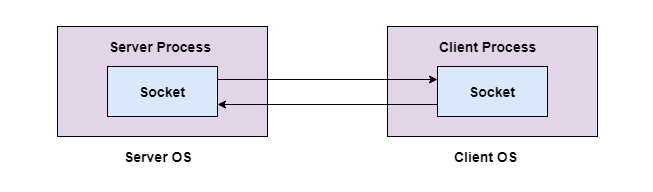


* + - **Socket**

Les sockets facilitent la communication entre deux processus sur la même machine ou sur des machines différentes. Elles sont utilisées dans un cadre client/serveur et se composent de l'adresse IP et du numéro de port. De nombreux protocoles d'application utilisent les sockets pour la connexion et le transfert de données entre un client et un serveur.

La communication par sockets est de très bas niveau, car les sockets ne font que transférer un flux d'octets non structuré entre les processus. La structure du flux d'octets est imposée par les applications client et serveur.

Voici un diagramme qui illustre les sockets

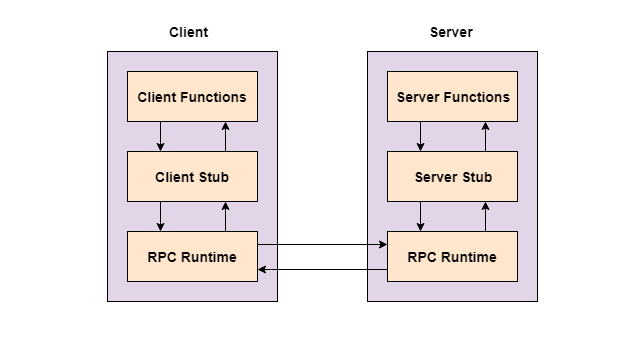


* + - **Appels procéduraux à distance (RPC)**

Il s'agit de techniques de communication interprocessus qui sont utilisées pour les applications client-serveur. Un appel de procédure à distance est également connu comme un appel de sous-routine ou un appel de fonction.

Un client a une demande que le RPC traduit et envoie au serveur. Cette demande peut être une procédure ou un appel de fonction vers un serveur distant. Lorsque le serveur reçoit la demande, il renvoie la réponse requise au client.

Voici un diagramme qui illustre les appels de procédure à distance



Les trois méthodes ci-dessus seront discutées dans des articles ultérieurs, car elles sont toutes assez conceptuelles et méritent leurs propres articles séparés.