Prosit Communication Interprocessus IPC

AIDE MEMOIRE IPC

Communication inter-processus (IPC) = Mécanismes permettant la communication entre processus sur une même machine voir entre hôtes d'un réseau. Communication permettant **échange de données** ou/et **synchronisation** entre les processus/threads/hôtes.
Ils s'appliquent à différents niveaux : **THREADS**, **PROCESSUS**, **MACHINES**

Processus = Processus lourd = Un programme en cours d'exécution par un ordinateur. De façon plus précise, il peut être défini comme : un ensemble d'instructions à exécuter, pouvant être dans la mémoire morte, mais le plus souvent chargé depuis la mémoire de masse vers la mémoire vive + un espace d'adressage en mémoire vive pour stocker la pile, les données de travail, etc. + des ressources telles que les ports réseau.

Thread = processus léger = Un thread ou fil (d'exécution) ou tâche est similaire à un processus car tous deux représentent l'exécution d'un ensemble d'instructions du langage machine d'un processeur. Du point de vue de l'utilisateur, ces exécutions semblent se dérouler en parallèle. Toutefois, là où chaque processus possède sa propre mémoire virtuelle, les threads d'un même processus se partagent sa mémoire virtuelle. Par contre, tous les threads possèdent leur propre pile d'appel.

Mécanismes de Synchronisation

Les mécanismes de synchronisation sont utilisés pour résoudre les problèmes de **sections critiques** et plus généralement pour bloquer et débloquer des processus suivant certaines conditions.

Exclusion mutuelle : ressource accessible par une seule unité à la fois (Ex : robinet)
Problème de cohorte : ressource partagée par au plus N utilisateurs (Ex : parking de 500 voitures)
Rendez-vous : des processus collaborant doivent s'attendre mutuellement
Producteurs/Consommateurs : un processus doit attendre la fin d'un autre
Lecteurs/Rédacteurs : notion d'accès exclusif entre catégories d'utilisateurs (Ex : Un fichier pouvant être lu par plusieurs, si personne ne le modifie)

Signaux **PROCESSUS**	Les signaux sont à l'origine destinés à tuer (terminer) un processus dans certaines conditions, par exemple le signal SIGSEGV tue un processus qui effectue un accès à une zone de mémoire qu'il n'a pas allouée. Les signaux peuvent cependant être déroutés vers d'autres fonctions. Le blocage d'un processus se fait alors en demandant l'attente de l'arrivée d'un signal et le déblocage consiste à envoyer un message au processus.
Sémaphore **THREADS**	Les sémaphores sont un mécanisme plus général, ils ne sont pas associés à un type particulier de ressource et permettent de limiter l'accès concurrent à une section critique à un certain nombre de processus. Pour ce faire les sémaphores utilisent deux fonctions : P et V, et un compteur. La fonction P décrémente le compteur, si le compteur est nul le processus est bloqué. La fonction V incrémente le compteur et débloque l'un des processus bloqué.
Verrou **PROCESSUS**	Les verrous permettent de bloquer tout ou une partie d'un fichier. Ces blocages peuvent être réalisés soit pour les opérations de lecture, soit d'écriture, soit pour les deux. Ils sont implémentés par le système de fichier.

Prosit Communication Interprocessus IPC

AIDE MEMOIRE IPC

Mécanismes d'échange de données

Mémoire partagée entre threads **THREADS**	Dans le cas de processus légers (thread en anglais), l'espace mémoire des processus est partagé, la mémoire peut donc être utilisée directement. Les échanges sont réalisés en plaçant les données en mémoire dans des variables partagées par les processus.
Mémoire partagée entre processus **PROCESSUS**	On utilise alors un mécanisme de partage de mémoire, tel que les segments de mémoire partagée dans Unix. Dans un contexte de la programmation concurrente, le partage de mémoire est un moyen de partager des données entre différents processus : une même zone de la mémoire vive est accédée par plusieurs processus. C'est le comportement de la mémoire de threads issus d'un même processus. Pour cela, dans un système utilisant la pagination, la table de page de chaque processus contient les pages mémoires communes, mais chaque processus ne les voit pas nécessairement à la même adresse.
Fichier **PROCESSUS/MACHINES**	Les fichiers peuvent être utilisés pour échanger des données entre plusieurs processus concurrents. Les processus voulant envoyer des données écrivent dans un (ou plusieurs) fichier(s) à certaines positions ; les processus souhaitant recevoir ces données se positionnent à ces positions dans le (ou les) fichier(s) et les lisent. Ce type d'échange est possible entre des processus concurrents locaux en utilisant le système de fichiers local, ou des processus concurrents distants en utilisant un système de fichiers distribué, tel que NFS.
Tube (pipe) **PROCESSUS**	En génie logiciel, un tube ou une pipeline est un mécanisme de communication inter-processus sous la forme d'une série de données, octets ou bits, accessibles en FIFO. Ces tubes sont détruits lorsque le processus qui les a créés disparait. Les tubes des shell, inventés pour UNIX, permettent de lier la sortie d'un programme à l'entrée du suivant. Exemple : `find –r readme | grep .txt` Les tubes peuvent être ouverts dans un programme informatique pour faire communiquer plusieurs processus. Sous unix le tube est implémenté grâce aux appels systèmes pipe(), fork() et exec().

Tube nommé (named pipe ou FIFO)
PROCESSUS
	En informatique, le terme tube nommé (calqué sur l'anglais named pipe) est une mise en œuvre des tubes Unix. Comme les tubes anonymes, les tubes nommés sont des zones de données organisées en FIFO mais contrairement à ceux-ci qui sont détruits lorsque le processus qui les a créés disparait, les tubes nommés sont liés au système d'exploitation et ils doivent être explicitement détruits.
	
	Exemple dans un shell unix :
	`mkfifo my_pipe`
	`rm my_pipe`
	Avec l'API windows en C++ :
	`HANDLE pipe = CreateFile(`
	`lpszPipename, // pipe name`
	`GENERIC_READ | // read and write access`
	`GENERIC_WRITE,`
	`0, // no sharing`
	`NULL, // default security attributes`
	`OPEN_EXISTING, // opens existing pipe`
	`0, // default attributes`
	`NULL); // no template file`

Fichier mappé en mémoire (memory mapped file) **PROCESSUS/THREADS**	Un fichier mappé en mémoire monte le contenu d'un fichier en mémoire virtuelle. Ce mappage entre un fichier et un espace mémoire permet à une application, y compris les processus multiples, de modifier le fichier en lisant et en écrivant directement dans la mémoire. Il existe deux types de fichiers mappés en mémoire :

	• Fichiers mappés en mémoire persistants • Fichiers mappés en mémoire non persistants
Clipboard **PROCESSUS**	En informatique, un presse-papier est une fonctionnalité qui permet de stocker des données que l'on souhaite dupliquer ou déplacer. Il utilise une zone de la mémoire volatile de l'ordinateur, pouvant contenir des informations de nature diverse (texte, image, fichier, etc.). Le format de transfert doit être géré par l'émetteur et le récepteur, au pire le contenu peut être converti en texte brut.
Spécifiques à Windows **PROCESSUS**	**OLE** : Object Linking and Embedding (OLE) (littéralement « liaison et incorporation d'objets ») est un protocole et un système d'objets distribués, mis au point par Microsoft. Il permet à des applications utilisant des formats différents de dialoguer. Par exemple, un traitement de texte peut insérer une image provenant d'un logiciel de traitement d'image. **Mailslot** : système de communique half-duplex où un espace de message est créé par un server, puis des clients pourront y écrire des messages. Méthode simple pour que des applications s'échangent des messages courts. Permet aussi d'envoyer un message à tous les ordinateurs dans un domaine réseau.

Mécanismes de synchronisation et d'échange de fichier

Socket **PROCESSUS/MACHINES**	Socket (en français interface de connexion) mécanisme par lesquelles une application peut se brancher à un réseau et communiquer ainsi avec une autre application branchée depuis un autre ordinateur. Il y plusieurs types de socket : • Internet : communication bidirectionnelle entre des processus qui sont sur des machines différentes sur un réseau IP. • Unix : les sockets du domaine UNIX utilisent le système de fichiers comme espace de noms. En plus d'envoyer des données, ces processus peuvent envoyer des descripteurs de fichiers sur un socket du domaine Unix, en utilisant les fonctions d'appel système « *sendmsg* » et « *recvmsg* ». • Socket brut (raw) : conçu manuellement (forgé). Les sockets raw sont nécessaires aux protocoles qui sont directement encapsulés dans IP, sans passer par TCP. On peut par exemple citer le protocole de routage dynamique OSPF, ainsi que le protocole ICMP utilisé par la commande ping
RPC **PROCESSUS/MACHINES**	En informatique et en télécommunication, RPC (Remote Procedure Call) est un protocole réseau permettant de faire des appels de **procédures** sur un ordinateur distant à l'aide d'un serveur d'applications. Ce protocole est utilisé dans le modèle client-serveur pour assurer la communication entre le client, le serveur et des éventuels intermédiaires. Ce système est également utilisé pour la conception des micro-noyaux. Secure RPC (S-RPC) protège les appels RPC avec un mécanisme d'authentification.
Message Queue **PROCESSUS/MACHINES**	Une file d'attente de message ou simplement file de messages (message queue) est une technique de programmation utilisée pour la communication interprocessus ou la communication de serveur-à-serveur. Les logiciels fournissant ce type de service font partie des « Message-Oriented Middleware » ou MOM.
Message Passing Interface (MPI) **PROCESSUS/MACHINES**	MPI (The Message Passing Interface) est une norme définissant une bibliothèque de fonctions, utilisable avec les langages C, C++ et Fortran. Elle permet d'exploiter des ordinateurs distants ou multiprocesseur par passage de messages. Il existe aussi des implémentations en Python, OCaml, Perl et Java. La dernière version est MPI-2.