DUT Informatique - Module M-4102C Modélisation et construction des applications réparties

Applications réparties (*distributed systems*)

J. Christian Attiogbé

Janvier 2017, maj 2020

Motivations

Exercices

- Construire un logiciel pour contrôler des robots coopérant dans une usine/maison/entreprise.
- Construire un logiciel pour gérer l'interaction entre des joueurs dans un logiciel de jeu.
- Construire un logiciel pour gérer l'interaction entre des applications qui partagent une ressource.
- Construire un logiciel qui gère l'interaction entre des applications qui à distance, les unes produisent des données et les autres, consomment les données.

Méthodes

Quelles méthodes ? quels modèles ?
Quelles technologies ? quels langages ? quelles plateformes ?

Motivations

- A travers cet enseignement,
  vous allez acquérir les connaissances de base pour
  comprendre et développer ce type de logiciel.
- Nous allons mettre en œuvre les compétences acquises sur des exemples et des cas d'étude.

Objectifs du cours

Objectifs

Ce cours a pour but de donner aux étudiants les méthodes et techniques de base pour la modélisation, la conception, et la réalisation de systèmes et logiciels répartis (*distributed systems*).

Moyens

Encadrement : CM 6h (4*1h20) - TD/TP 17,5h (7 sem., 2*1h20/sem)
Prérequis : Systèmes M3101, Réseaux M3102
Fil conducteur : applications réelles à réaliser au fil des TDs

Intervenants

Christian Attiogbé (CM, TD) & Loïg Jezequel (TD)

Plan du cours

Exemples d'applications réparties

Figure: Poker en réseau : application répartie

(source http://www.xoen.org)

Exemples d'applications réparties

- Serveur FTP et clients FTP (comme HTTP)
- Système de Chat sur le réseau Internet.
- Système de réservation de vols/trains...
- Contrôle de processus industriels
- Systèmes de gestion de versions (svn, ...)
- Systèmes de bases de données réparties
- Serveurs de fichiers (ou autres ressources) à des clients.
- le DNS (Domain Name System)
- le World Wide Web

Systèmes et applications répartis

Application répartie

Une application répartie = des procédures réparties sur un réseau et interagissant sur des données réparties, afin d'effectuer une tâche globale mais spécifique ; donc il y a dépendance entre les procédures.

Interaction

Interaction = communication dans le temps par des échanges entre des entités (processus), et synchronisation à certains moments.

Système (d'exploitation) réparti

Des procédures réparties pour la gestion des ressources communes et de l'infrastructure, liées au matériel (processeur).

Rappel de la structuration d'un hôte (machine) ouvert

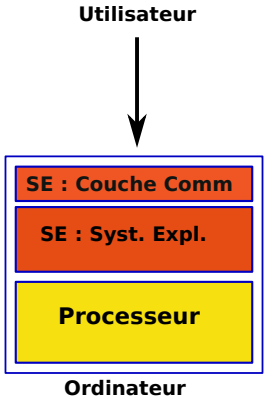


Figure: Couches au niveau d'un ordinateur

Infrastructure réseau (et archi. des applis réparties)

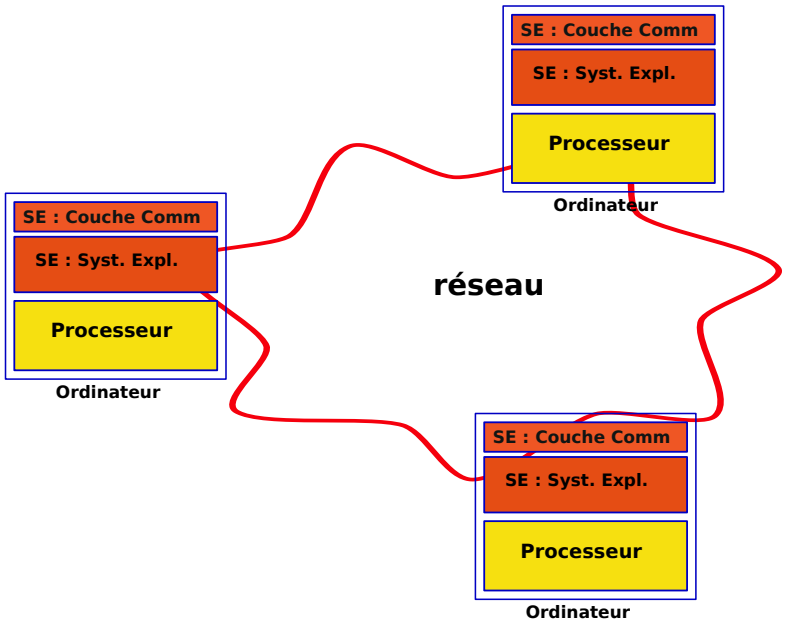


Figure: Couches au niveau d'un ordinateur dans le réseau

Utilisateur et structure du réseau

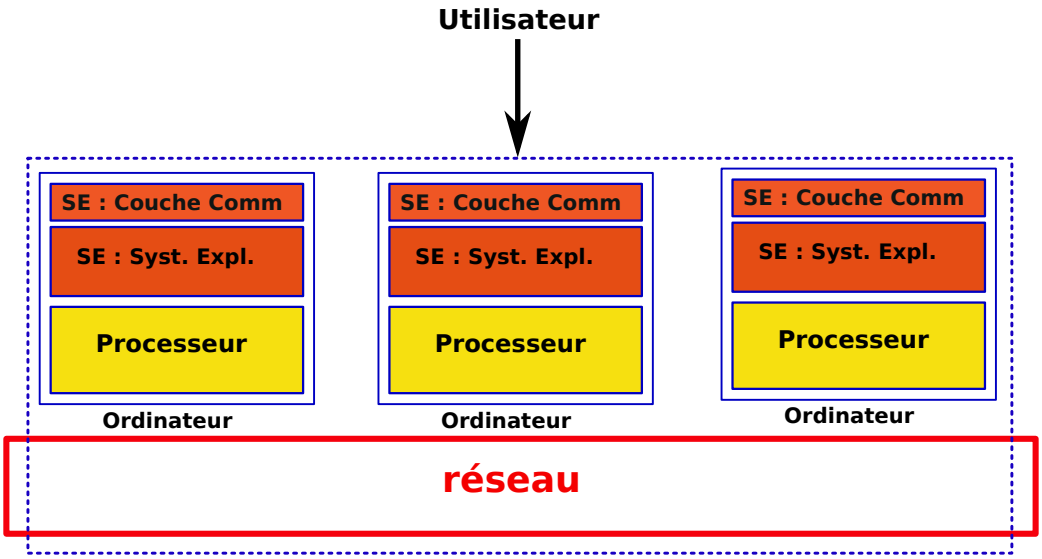


Figure: Couches au niveau réseau

Structure en couche des applications réparties

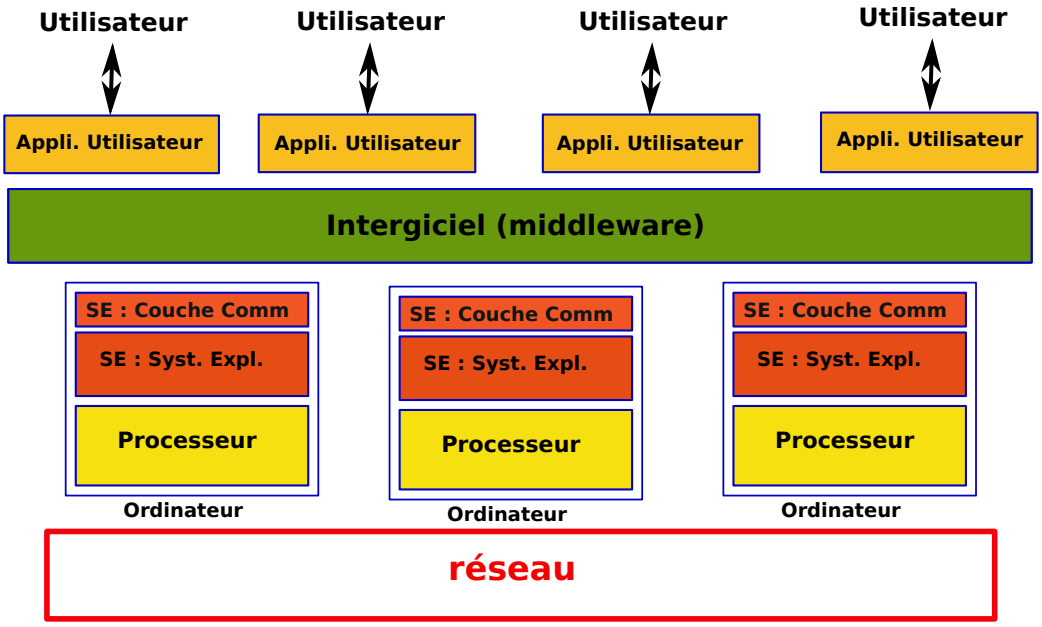


Figure: Couches dans une application répartie

Echange entre des applications réparties

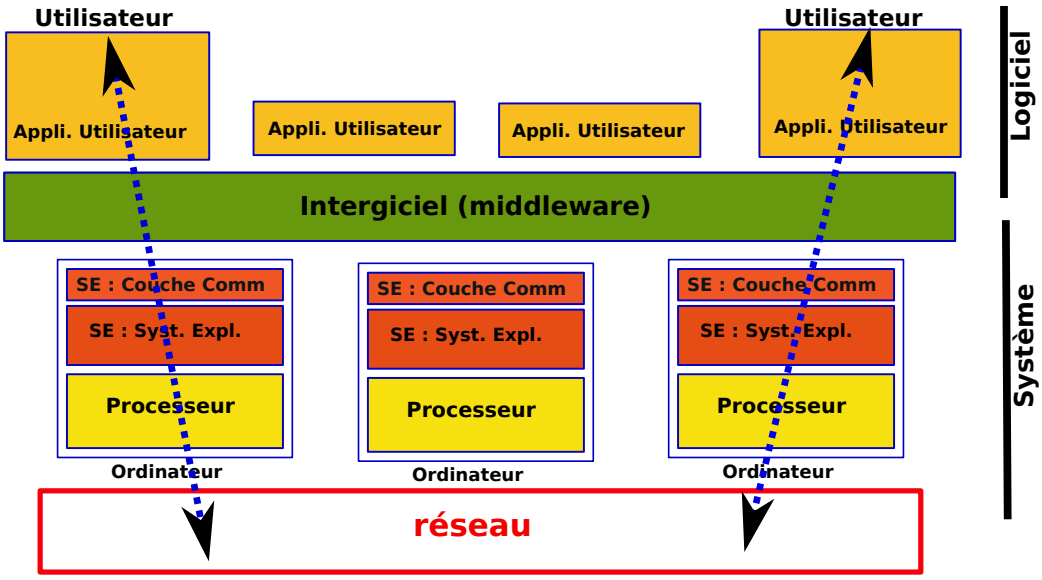
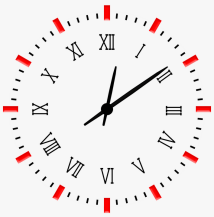


Figure: Flux des échanges dans une application répartie

Caractérisation des communications : mode et type



Modes de communication

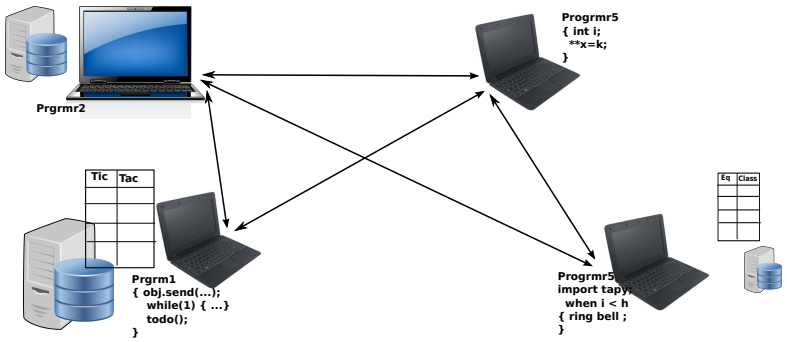
- mode connecté (*connected*)
- mode non connecté (*unconnected*).

Tyes de communication

- communication de type synchrone (*synchronous*) vs
- communication de type asynchrone (*asynchronous*)

Difficultés de la programmation répartie

Les systèmes ou applications réparties sont souvent très complexes du fait des caractéristiques de leur environnement :
le réseau et ses caractéristiques, l'interaction avec les processus, les accès aux données, la synchronisation, l'asynchronisme, etc.



Asynchrone : pas d'horloge globale (chaque entité à son horloge) + pas de synchronisation des échanges.

Difficultés de la programmation répartie

Ne pas faire les hypothèses non rigoureuses ou erronnées suivantes :

- le réseau est fiable et sûr NON !
- le réseau est homogène, statique, ne change pas de structure et de caractéristiques NON !
- Il n'y a pas de temps de latence entre les processus, les échanges sont immédiats (pas de coût de transport) NON !
- La bande passante est large et constamment disponible NON !
- Il y a un moniteur/superviseur NON !
- ...

Difficultés de la programmation répartie

- Désignation et liaison des entités
- Transmission des données entre les entités
- Conception globale
- Changement continuel de la structure/composition du système global (+réseau)
- Pas d'hypothèse de fiabilité : gérer les pannes/défauts/fautes
- Propriété d'asynchronisme du système de communication (pas de borne supérieure stricte pour le temps de transmission des messages. Cela complique la détection des défaillances.)
- ...

Difficultés liées à l'algorithmique distribuée : exploitation de connaissances partielles, communications asynchrones et consensus.

Propriétés attendues d'un système réparti

(pas seulement pour les OS, mais aussi pour les applications réparties)

- Disponibilité : Le système doit pouvoir fonctionner (au moins de façon dégradée) même en cas de défaillance de certains de ses éléments
- Tolérance aux pannes : le système doit pouvoir résister à des perturbations du système de communication (perte de messages, déconnexion temporaire, performances dégradées)
- Absence de blocage
- Transparence des accès, de la localisation des entités de la concurrence, du passage à l'échelle, de la mobilité... (ISO)

Autres prioriétés à connaître et traquer/analyser

- *Race condition* : situation de concurrence d'accès concurrent à une ressource ; quel résultat d'exécution ? (non prévisible !)
- La bornitude : absence d'évolution de tout ou partie du système quand un certain état est atteint
- Le non-déterminisme : effet des actions non prévisible selon les exécutions
- L'équité : tous les processus ont les mêmes chances d'évolution par rapport à l'usage des ressources (pas plus/moins de priorité)
- ...

Concepts et principes de base de la répartition

Le passage de message (*message passing*) est un mécanisme fondamental dans les systèmes répartis ; c'est aussi le mécanisme ou modèle le plus simple.

- Un processus envoie un message correspondant à une requête
- Le message est délivré à son destinataire, qui le traite et envoie une réponse en retour
- La réponse peut provoquer une autre requête, suivie d'une autre réponse, ainsi de suite.

Un exemple typique de cette interaction est la manière dont un processus client (un utilisateur avec un navigateur) communique avec un serveur web (serveur de données) — à travers le réseau.

Répartition des données et des traitements

- Les données ou les traitements peuvent être répartis.
- Les données réparties peuvent être accédées par des processus différents (donc *race condition*).

  Pour traiter le problème des accès concurrents :
  - exclusion mutuelle, section critique

  Solution bas niveau :
  - mécanismes de verrous et de synchronisation
- Les traitements répartis interagissent via des messages ou des données partagées (en mémoire -virtuelle- commune).
- Mécanismes de haut niveau : langages+librairies, intergiciels (*middlewares*)

Répartition : abstractions pour les applications

Du haut niveau vers le plus bas niveau d'abstraction

Espace des données *object space*
Services réseaux, (*broker*) de requêtes, agents mobiles *network services, object request broker, mobile agent*
Appel de procédures à distance, invocation de méthodes à distance *remote procedure call (**RPC**) , remote method invocation (**RMI**)*
Client - Serveur
Passage de message *message passing*

(source : Mei-Ling Liu, Dept. Computer Sc. Cal Poly, San Luis Obispo, USA)

Des modèles de mise en œuvre de la répartition

Il y a plusieurs modèles de la répartition :

☞ Modèle de communication par messages
Modèle de communication par événements (*event-based technology*)
☞ Modèle Client-Serveur : RPC (1988), CORBA (1991), RMI (1996), ...
☞ Modèle des objets répartis CORBA, RMI
Modèle à base de composants : Bean, Enterprise Java Beans, CORBA, ...
Modèles à mémoires virtuelles partagées : Modèle basé sur les tuples (*tuple-based technology*)
...

On peut les organiser par niveaux d'abstraction.

Modèle à échange de messages (*Message passing*)

Un message = des requêtes ou réponses avec ou sans données.

- Les entités qui communiquent échangent des messages ;
- les entités se synchronisent via les messages pour l'accomplissement des traitements.
- L'échange de messages peut se faire
  - en point à point (un émetteur-un récepteur) ou
  - en multi-points (des récepteurs s'abonnent) - **Publish-Suscribe**

Modèle à échange de messages (*Message passing*)

- Basé sur un mode de communication **asynchrone**.
- Processus Client et serveur ne s'attendent pas.
- Les messages sont stockés dans une queue et traités ensuite.
- Les intergiciels (*middleware*) assurent une partie de la fiabilité.

Type de communication (en *Message passing*)

- Communication temporaire ou évanescent (*Transient*) : messages non stockés dans des registres ; le message est détruit si son récepteur n'est pas en exécution (exemple avec les sockets).
- Communication persistante (messages stockés dans des registres et délivrés de façon différée).

Boîtes au lettres ou files d'attentes pour les messages échangés.

Le principe du *Message Passing*

Les opérations élémentaires requises pour mettre en œuvre le passage de messages sont : *send*, *receive*.
Selon le mode de communication entre les entités (mode connecté, ou non-connecté), des opérations telle que *connect* et *disconnect* sont requises.

Mise en œuvre avec les socket.

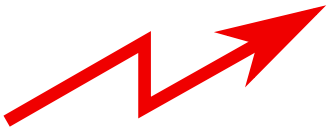
- L' interface de programmation SOCKET est basée sur ce paradigme.
- les processus communicants/répartis effectuent des entrées/sorties à un niveau d'abstraction qui masque le réseau de communication et le système d'exploitation.
- Les API SOCKET dans les langages, constituent un outil pour mettre en œuvre l'interaction entre les processus.

Référence : `http://www.sockets.com/`

Modèle de communication par événements

Concepts de bases

- événements,
- réactions,
- liaison entre événement et réaction.



- Communication ouverte/anonyme:
  indépendance entre l'émetteur et les consommateurs d'un événement.
- Deux modes : PULL et PUSH
  - En mode PULL : les clients vont chercher les messages.
  - En mode PUSH : le client a une méthode dédiée à chaque type de message ; cette méthode est appelée à chaque occurrence de l'événement.

Modèle Client/Serveur

Des processus et des ressources convoitées

- Le serveur expose ses services ;
  il attend des requêtes des clients et leur répond.
- Le client émet des requêtes vers les serveurs ;
  interagit avec le serveur

- Modèle basé sur une communication synchrone.
- Le client reste en attente de la réponse du serveur (risques de blocages).
- N'est pas approrié à certains types d'applications

Mise en œuvre du modèle client/serveur

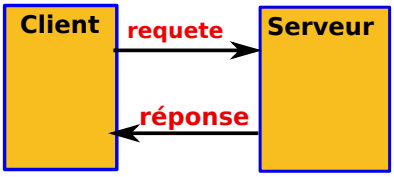
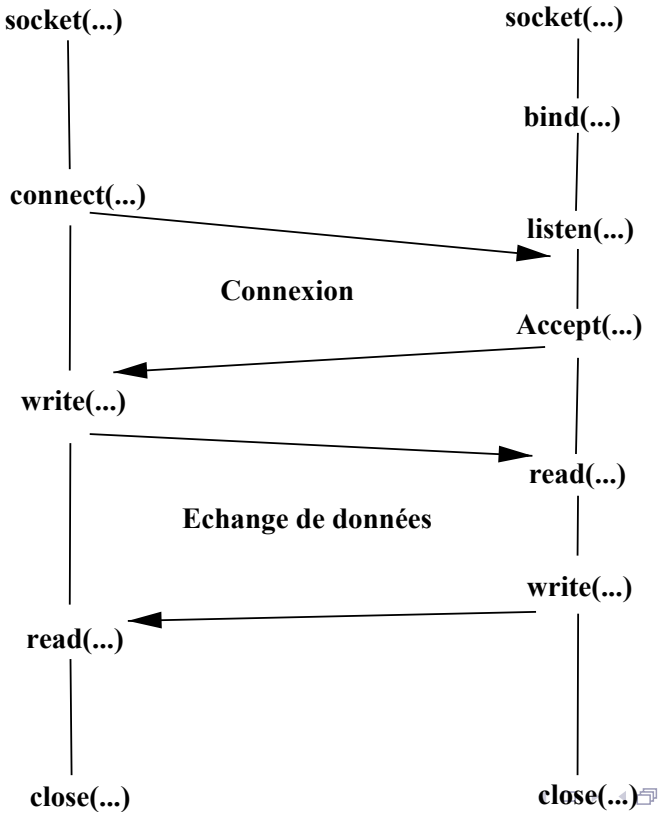


Figure: Modèle client-serveur

- Par mécanismes de bas niveau :
  - les sockets (en mode connecté ou non connecté)
- Par des mécanismes de haut niveau :
  - les intergiciels,
  - dans un langage dédié avec des RPC,
  - avec des objets répartis (création des objets et appels de méthodes à distance)

Schéma de principe des programmes Client/Serveur



Modèle Pair à pair (*Peer to peer*)

Le modèle Client/Serveur se généralise en une appoche ou chaque hôte joue à la fois le rôle de client et de serveur.
Les systèmes pair à pair (*Peer to peer*, P2P) sont fondés sur ce modèle. Ils permettent de partager des données à très grande échelle.

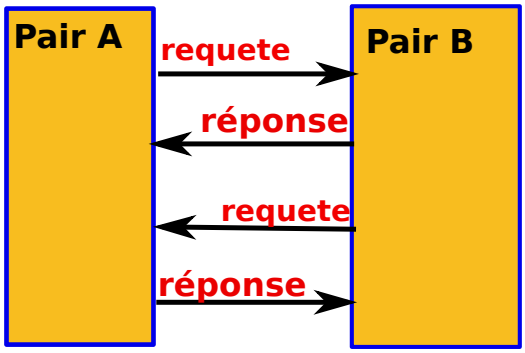


Figure: Modèle Pair-à-pair

Evolutions du Modèle Client/Serveur

Evolutions du modèle client/serveur

- Client/Serveur traditionnel : RPC (1988)
- Client/Serveur basé sur les objets :
  CORBA (1991), RMI(1996), DCOM, ...
- Client/Serveur basé sur le Web :
  CGI, Servlet, asp, jsp, php, ...

Appel de procédure à distance (*Remote Procedure Call: RPC*)

- L'appel de procédure est un mécanisme de base en programmation (modulaire), 1988
- Ce mécanisme est étendu pour appeler des procédures situées à distance.
- Implanté par SUN en 1988, mais proposé par par Birel et Nelson en 1984.
- Les procédures peuvent être paramétrées ; les valeurs sont transmises, exécutées à distance, puis le résultat transmis à l'appelant (en *marshalling* ou en *parameter passing*).

Mise en œuvre du RPC

Marshalling de paramètres

client et serveur se mettent d'accord sur la représentation des données

- *Wrapping* des paramètres = les transformer en valeurs (octets).
- Nécessite un accord sur la représentation machine-indépendante des données.

Passage des paramètres

- Pas de passage des valeurs mais des références des paramètres
- Nécessite un mécanisme d'accès distant aux données réparties (type CORBA)

Objets répartis (*distributed objects*)

Un objet réparti est un objet résidant sur une machine et dont les méthodes d'accès peuvent être évoquées à partir de traitements situés sur d'autres machines.

Principe de la technologie des objets répartis

- intercepter les appels aux objets répartis, et
- exécuter des services-systèmes dédiés à la localisation des objets puis l'envoi de données et des requêtes/instructions.

Le réseau et les communications sont complètement masqués aux programmeurs.

Exemples

CORBA, basée sur une approche multi-langage.
RMI, basée sur JAVA.

Modèle à base de composants

Composant

Un composant = entité logicielle autonome réutilisable (à travers des interfaces publics)

Exemples de composants : Beans, EJB, CORBA Component

Application répartie

Une application (répartie) = assemblage de plusieurs composants (répartis)

Les intergiciels mettent en œuvre l'interaction entre les composants.

Modèle à mémoire virtuelle partagée

- On définit des espaces mémoire communs (*share memory*) pour les communications entre les processus.
- Les applications accèdent aux espaces virtuels, par des appels/méthodes.
- Les processus/applications se synchronisent via des variables partagées.
- Nécessite une mise en œuvre rigoureuse et efficace.

Exemples

Modèles à espace de tuples comme dans Linda (David Gelernter), ou comme dans les bases de données relationnelles partagées.

Algorithme d'exclusion mutuelle

Mécanisme fondamental : exclusion mutuelle
Imaginons une section critique, convoitée par n processus
donc *race condition*

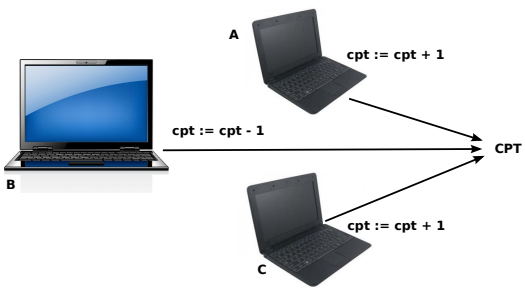


Figure: Concurrence d'accès

La modification de la variable doit se faire en section critique

Algorithme d'exclusion mutuelle

Chaque processus peut :

- demander à entrer en SC
- attendre la/les réponses à sa demande de SC
- recevoir une demande d'entrée en SC
- répondre à une demande
- entrer en SC
- sortir de la SC

Mais les processus sont distants, séparés par le réseaux, simultanés, ...
L'algo assure la synchronisation dans les systèmes répartis

Exclusion mutuelle : un algorithme réparti (distribué)

Lamport 1978, Ricart et Agrawal 1981
L'algorithme demande l'existence d'un ordre total sur tous les événements du système.
Quand un processus veut entrer en section critique :

- il construit un message contenant :
  - le nom de la section critique dans laquelle il veut entrer,
  - son numéro de processus et
  - l'heure courante (estampille).
- Il envoie ce message à tous les autres processus + lui même.

En supposant que l'émission des messages est fiable, chaque message est acquitté.

Exclusion mutuelle : un algorithme distribué

Quand un processus reçoit un message *demande d'entrée en SC* :

- il prend une décision qui dépend de son état et de la section critique sollicitée par le message.
  1. si le récepteur n'est pas dans la section critique et ne veut pas entrer dans celle-ci, il retourne simplement un message OK à l'emetteur ;
  2. si le récepteur est déjà dans la section critique, il ne répond pas, il mémorise la demande dans une file d'attente ;
  3. si le récepteur veut entrer en section critique, mais ne l'a pas encore fait, il compare l'estampille du message venant d'arriver à celle contenue dans le message qu'il a envoyé aux autres. La plus ancienne gagne. Si le message est plus vieux, le récepteur répond par un message OK. Si c'est l'estampille de son message qui est la plus ancienne, le récepteur mémorise la demande et n'envoie rien.

Exclusion mutuelle : un algorithme réparti (distribué)

Après avoir envoyé ses demandes d'entrée en section critique, un processus (demandeur) se met en état d'attente jusqu'à ce que tous les autres lui donnent leur autorisation.

Dès que toutes les autorisations sont accordées, il peut entrer en section critique.

Quand il sort, il envoie un message OK à tous les processus présents dans la file d'attente et les supprime tous de la file.

Dans les conditions normales, cet algorithme fonctionne correctement.

Que se passe-t-il quand une machine est en panne ?

Exclusion mutuelle : un algorithme réparti (distribué)

L'algorithme est distribué ; toutes les machines y sont impliquées.
Quand une machine est bloquée, tout le reste est bloqué.

Le problème de l'algorithme

S'il y a n ordinateurs, il y a n points faibles (contrairement à un algorithme centralisé ou il y aurait 1 point faible).

Solution

- Quand un message arrive, il doit être acquitté : accord ou refus.
- L'émetteur utilise un temporisateur et arme un délai quand il envoie une demande vers les autres machines.
- Si au bout du délai de temporisation il n'a pas obtenu d'acquittement, le message est déclaré perdu et l'algorithme peut fonctionner sans la machine concernée.

Conclusion sur l'exclusion mutuelle

- On sait écrire des algorithmes répartis
- Ils sont plus difficiles à élaborer et à prouver
- Ils sont mis en œuvre dans les systèmes

Assistance au développement

- Décharger le programmeur des tâches de bas niveau, pour écrire les applications réparties
- Proposer des mécanismes et outils de plus haut niveau pour interagir avec l'environnement : les *middleware*.

Les *middleware* ou intergiciels

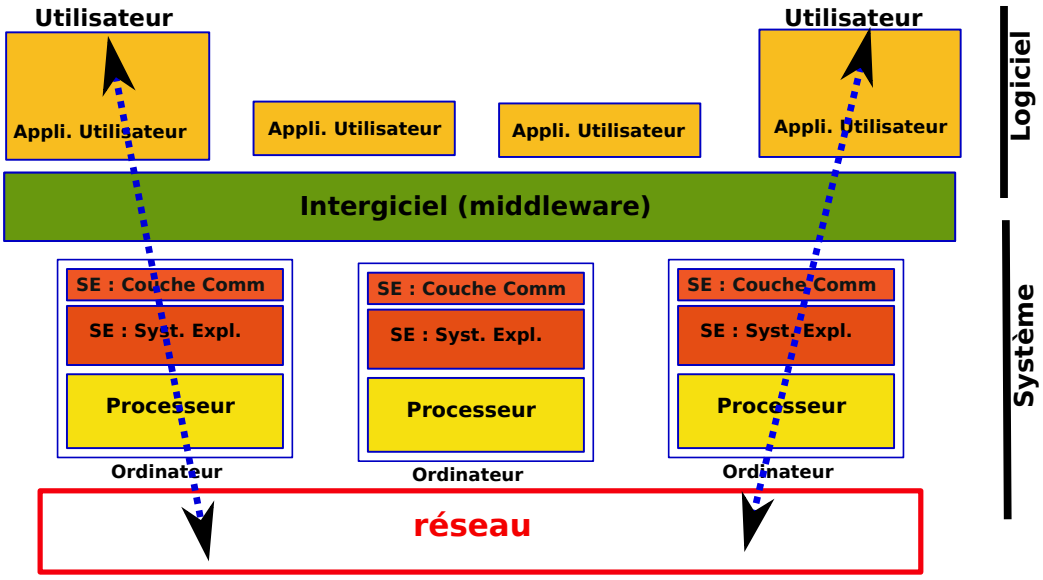


Figure: Application répartie et *middleware*

Les *middleware* ou intergiciels

Middleware

couche de logiciels pour :
i) masquer l'hétérogénéité des machines et systèmes
ii) masquer la répartition des traitements et données
iii) fournir une interface pratique et commune aux différentes applications (modèle de programmation + API)

Fonctionnalités offertes par un *middleware*

Communication, localisation, transactions, sécurité, administration.

Problèmes résolus par un *middleware*

intégration et interopérabilité =
indépendance entre les applications et le système d'exploitation, portabilité des applications, partage des services distribués.

Catégories et exemples de *middleware* répandus

- A base d'objets répartis (années 1990+) :
  CORBA (début 90), Java RMI(mi 90), DCOM (fin 90)
- A base de composants répartis (autour de 2000):
  Java Beans, Enterprise Java Beans (évol. de Java RMI), .NET, CCM (évol. de Corba)
- A base de Messages (Message-Oriented Middleware, MOM) :
  Message Queues (IBM, Microsoft), Publish-Subscribe
- Middlewares orientés SGBD
  ODBC, JDBC
- Pour l'intégration d'applications :
  Web Services (XML-RPC, *Simple Object Access protocol* (SOAP))

Les *middleware* basés sur les messages (MOM)

Ils assurent de façon fiable (en tolérant des défauts) une communication persistante en mode asynchrone

- Les messages envoyés entre les processus communicants sont enregistrés (queue de messages en attente)
- Les processus sont indépendants et ne se bloquent pas mutuellement.

Références

- A.S. Tannenbaum, *Distributed Operating Systems* Prentice Hall.
- Sacha Krakowiak, Systèmes et Applications Réparties, Université Joseph Fourier
- Martin Quinson, Cours de Programmation d'Applications Réparties
- A.D. Birell and B.J. Nelson, *Implementing remote procedure calls* ACM Trans. on Comp. Syst., vol. 2(1), 1984.
- Satyanarayanan, H. Siegel *Parallel communication in a large distributed environment*, ACM Trans. On Comp, vol 39(3), 1990

Conception et architecture des applis : multi-niveaux

- Deux niveaux : Client et serveur
  - les clients (dits lourds) traitent une partie des données et s'occupent aussi de la présentation des données.
  - Le serveur (a et) sert les données.

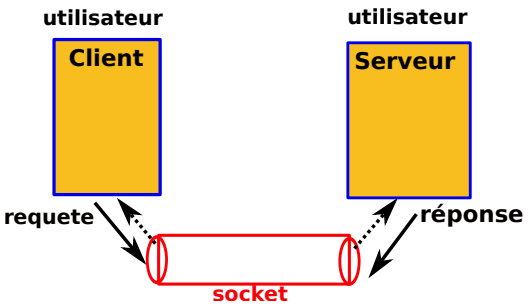


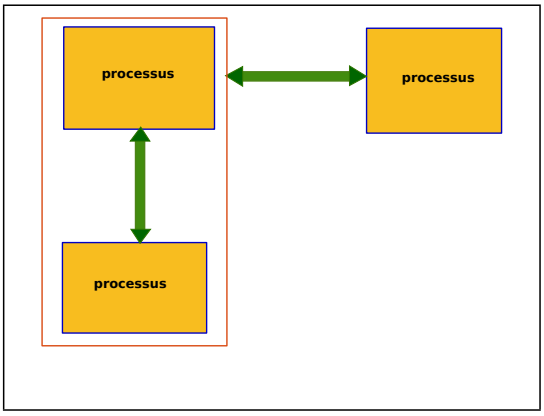
Figure: Modèle Client-Serveur

Imaginez la situation avec plusieurs clients.

Conception et architecture des applis : multi-niveaux

- Trois niveaux et plus
  - le client (dit léger) n'assure que la présentation des données ;
  - un serveur applicatif (commun aux clients) traitent les données ;
  - un serveur de donnés (délivre/stocke les données).

Conception et architecture des applis



- Abstraction des sous-systèmes, par leurs comportements : automates, etc
- Composition graduelle et analyse des comportements
- Réalisation (via des langages, middleware, ...)

⇒ Modélisation avec les Réseaux de Petri

Références

- A.S. Tannenbaum, *Distributed Operating Systems* Prentice Hall.
- Sacha Krakowiak, Systèmes et Applications Réparties, Université Joseph Fourier
- Martin Quinson, Cours de Programmation d'Applications Réparties
- A.D. Birell and B.J. Nelson, *Implementing remote procedure calls* ACM Trans. on Comp. Syst., vol. 2(1), 1984.
- Satyanarayanan, H. Siegel *Parallel communication in a large distributed environment*, ACM Trans. On Comp, vol 39(3), 1990