Systèmes concurrents

Philippe Quéinnec

ENSEEIHT
Département Sciences du Numérique

16 septembre 2020

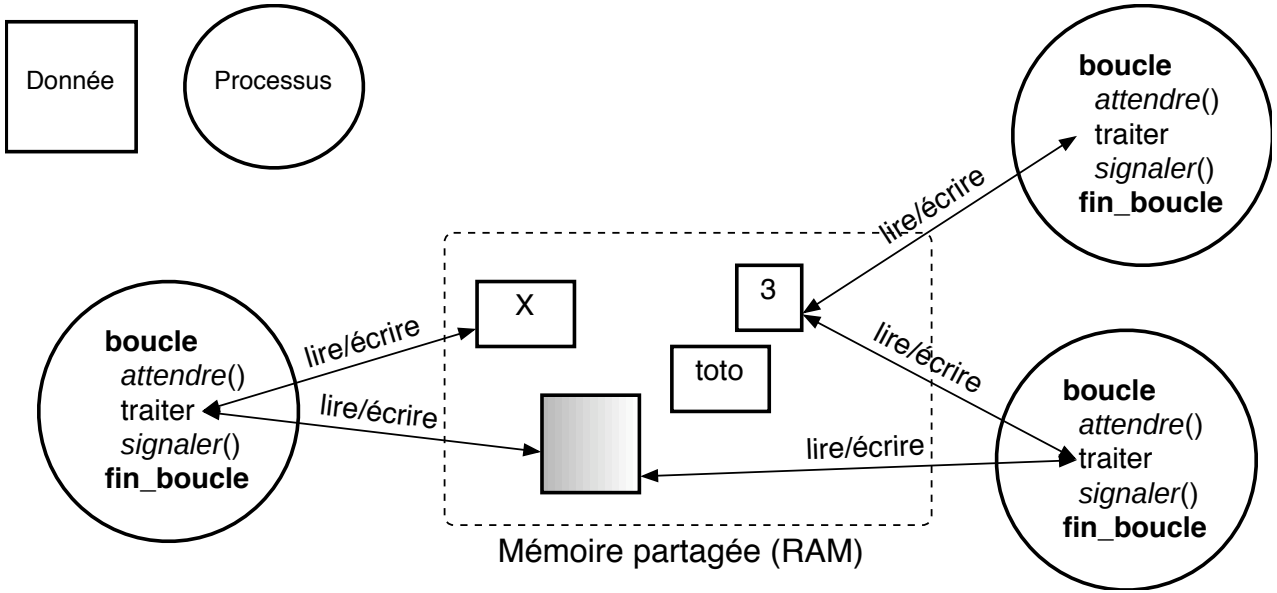
Septième partie

Processus communicants

Contenu de cette partie

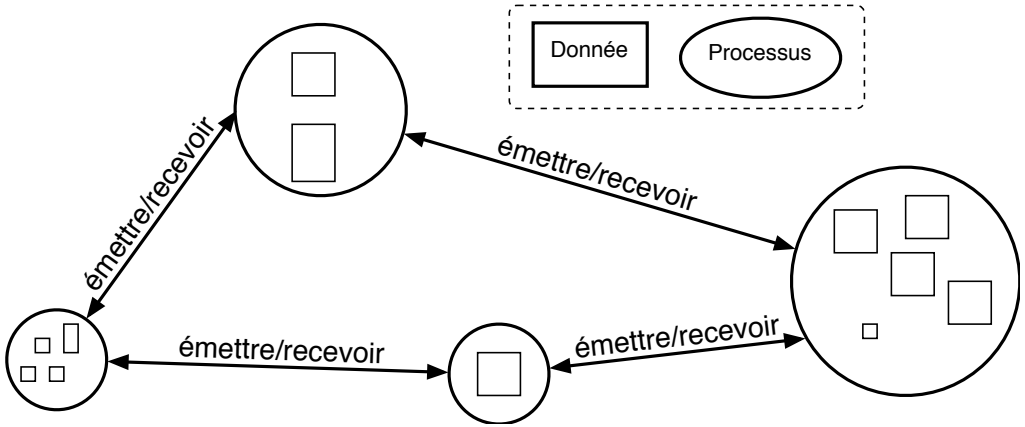
- Modèles de programmation concurrente
- Modèle des processus communicants
- Approche CSP/Go pour la programmation concurrente
  - Goroutine et canaux
  - Communiquer explicitement plutôt que partager implicitement
- Approche Ada pour la programmation concurrente
  - Tâches et rendez vous
  - Démarche de conception d'applications concurrentes en Ada
    - Transposition de la démarche vue dans le cadre de la mémoire partagée (moniteurs)
    - Extension tirant parti des possibilités de contrôle fin offertes par Ada

Modèles d'interaction : mémoire partagée ♪♪♪



- Données partagées
- Communication implicite
  - résulte de l'accès et de la manipulation des variables partagées
  - l'identité des activités n'intervient pas dans l'interaction
- Synchronisation explicite (et nécessaire)
- Architectures/modèles cibles
  - multiprocesseurs à mémoire partagée,
  - programmes multiactivités

Modèles d'interaction : processus communicants ♪♪♪



- Données encapsulées par les processus
- Communication nécessaire, explicite : échange de messages
  - Programmation et interactions plus lourdes
  - Visibilité des interactions → possibilité de trace/supervision
  - Isolation des données
- Synchronisation implicite : attente de message
- Architectures/modèles cibles
  - systèmes répartis : sites distants, reliés par un réseau
  - moniteurs, CSP/Erlang/Go, tâches Ada

Plan

Processus communicants

Principes

- Communication inter-processus avec des opérations explicites d'envoi / réception de messages
- Synchronisation via ces primitives de communication bloquantes : envoi (bloquant) de messages / réception bloquante de messages

- Communicating Sequential Processes (CSP) / Calculus of Communicating Systems (CCS) / π-calcul / Erlang / Go
- Ada

Les principes détaillés des échanges et leur utilisation pour développer des applications sont vus dans le module « intergiciels ». On ne s'intéresse ici qu'à la synchronisation.

Quelle synchronisation ?

Réception

Réception bloquante : attendre un message

Émission

- Émission non bloquante ou asynchrone
- Émission bloquante ou synchrone : bloque jusqu'à la réception du message = rendez-vous élémentaire entre l'activité émettrice et l'activité destinataire
- Rendez-vous étendu : bloquant jusqu'à réception + réaction + réponse ≈ appel de procédure

- Émission asynchrone ⇒ buffers (messages émis non reçus)
- Synchrone ⇒ 1 case suffit

Désignation du destinataire et de l'émetteur ♪♪♪

Nommage

- Direct : désignation de l'activité émettrice/destinataire

```
SEND message TO processName
RECV message FROM processName
```

- Indirect : désignation d'une boîte à lettres ou d'un canal de communication

```
SEND message TO channel
RECV message FROM channel
```

Multiplicité

$1-1$

Désignation de l'activité : 1 émetteur / 1 récepteur désignés

$n-1$

Canal réservé en lecture (consommation) : envoi par n'importe quelle activité ; réception par une seule, propriétaire du canal

$n-m$

Canal avec envoi par n'importe qui, réception par n'importe qui :

- pas de duplication : un seul destinataire consomme le message
- ou duplication à tous les destinataires (diffusion)

En mode synchrone, la diffusion est complexe et coûteuse à mettre en œuvre (nécessite une synchronisation globale entre tous les récepteurs)

Alternative

Alternative en émission ou en réception = choix parmi un ensemble de communications possibles :

```
RECV msg FROM channel1 OR channel2
(SEND msg1 TO pid1) OR (SEND msg2 TO pid2)
(RECV msg1 FROM channel1) OR (SEND msg2 TO channel2)
```

- Si aucun choix n'est faisable ⇒ attendre
- Si un seul des choix est faisable ⇒ le faire
- Si plusieurs choix sont faisables ⇒ sélection non-déterministe (arbitraire)

Divers

Émission asynchrone ⇒ risque de buffers pleins ♪♪♪

- perte de messages ?
- ou l'émission devient bloquante si plein ?

Émission non bloquante → émission bloquante ♪♪♪

introduire un acquittement

```
   (SEND m TO ch; RECV _ FROM ack)
 ‖ (RECV m FROM ch; SEND _ TO ack)
```

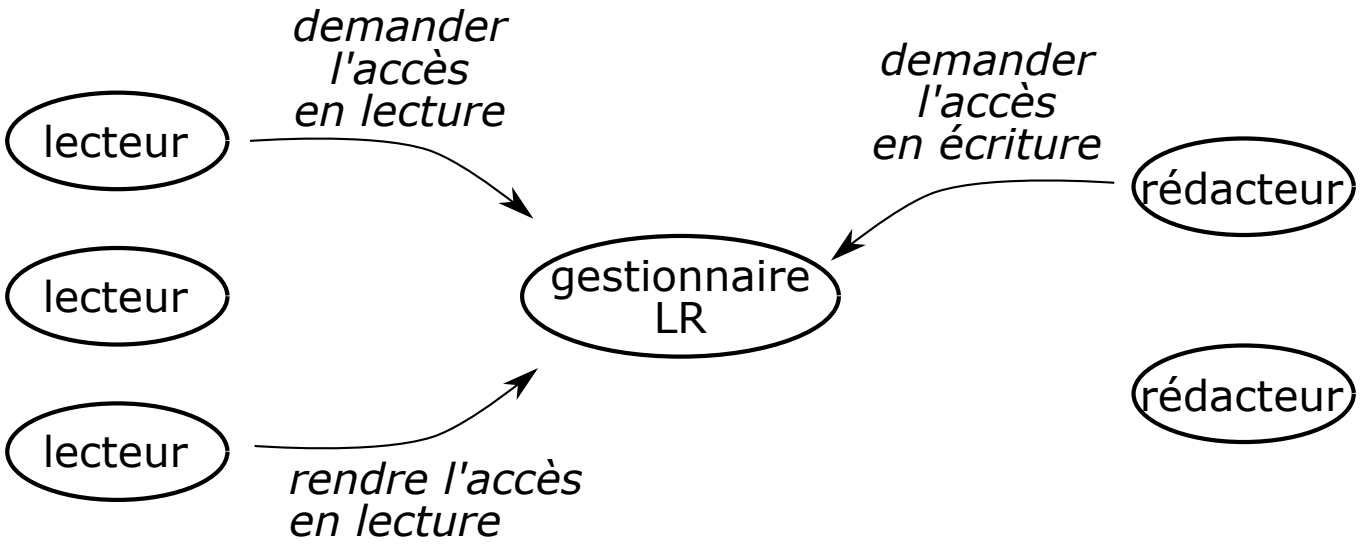
Émission bloquante → émission non bloquante ♪♪♪

introduire une boîte intermédiaire qui accepte immédiatement tout message et le stocke dans une file.

```
   (SEND m TO ch1)
 ‖ boucle (RECV m FROM ch1; insérer m dans file)
 ‖ boucle (si file non vide alors extraire et SEND TO ch2)
 ‖ (RECV FROM ch2)
```

Architecture

La résolution des problèmes de synchronisation classiques (producteurs/consommateurs. . .) ne se fait plus en synchronisant directement les activités via des données partagées, mais indirectement via une activité de synchronisation.



Activité arbitre pour un objet partagé

Interactions avec l'objet partagé

Pour chaque opération $\mathcal{O}p$,

- émettre un message de requête vers l'arbitre
- attendre le message de réponse de l'arbitre

($\Rightarrow$ se synchroniser avec l'arbitre)

Schéma de fonctionnement de l'arbitre

- L'arbitre exécute une boucle infinie contenant une alternative
- Cette alternative possède une branche par opération fournie
- Chaque branche est gardée par la condition d'acceptation de l'opération (suivie de l'attente du message correspondant)

Note : en communication synchrone, on peut se passer du message de réponse s'il n'y a pas de contenu à fournir.

Intérêt

+ découplage entre les activités clientes : l'interface partagée est celle de l'activité de synchronisation
+ réalisation centralisée et répartie
+ transfert explicite d'information : traçage
+ pas de données partagées $\Rightarrow$ pas de protection nécessaire
+ contrôle fin des interactions
+ schéma naturel côté client : question/réponse = appel de fonction
– multiples recopies (mais optimisations possibles)
– parallélisation du service : au cas par cas