Processus concurrents et parallélisme

Chapitre 5 - Communication inter-processus

Gabriel Girard

17 octobre 2022

Chapitre 5 - Communication inter-processus

Chapitre 5 - Communication inter-processus

Présentation

- Deux techniques de communications :
  - mémoire partagée
  - passage de messages

Chapitre 5 - Communication inter-processus

Mémoire commune

- Le programmeur est responsable de fournir la communication
- Nécessite de la synchronisation explicite

Chapitre 5 - Communication inter-processus

Passage de messages

- Les processus ne partagent pas de mémoire
- Doivent utiliser des appels systèmes pour communiquer

Passage de messages

- Primitives des base

  `send` *message* `to` *destination*

  `receive` *message* `from` *source*

- Autres primitives possibles

  `wait`

  `sendReply`

  `receiveReply`

Passage de messages

- Avantages $\rightarrow$ aucune synchronisation explicite
- Problèmes à résoudre :
  - Comment spécifier les noms de la source et de la destination ?
  - Comment synchroniser la communication ?

Canal de communication

- La désignation de la source et de la destination sert à établir un lien logique appelé canal de communication
- Un canal est représenté par un couple «(source,destination)»
- Technique de spécification du canal
  - la désignation directe
  - le port global (boîte aux lettre)
  - le port

La désignation directe

- La plus simple
- On utilise les noms des processus comme source et destination

`send` *message* `to` *processus*1
`receive` *message* `from` *processus*2

La désignation directe

- Propriétés du canal
  - un canal par paire de processus
  - un canal est associé à seulement deux processus
  - le canal est bidirectionnel

Exemple : système batch

```
Program OPSYS ;
process lecteur ;
        var ligne : entree ;
        loop
                lecture ligne ;
                send ligne to traitement ;
        end ;
end process ;
Process traitement ;
        var     ligne : entree ;
                resultat : sortie ;
        loop
                receive ligne from lecteur ;
                traitement de ligne et génération de resultat ;
                send resultat to imprimante ;
        end ;
end process ;
```

Exemple : système batch

```
Process imprimante ;
        var resultat : sortie ;
        loop
                receive resultat from traitement ;
                impression de resultat ;
        end ;
end process ;
```

La désignation directe

- Cet exemple illustre le concept de pipeline
- La désignation directe est bien adaptée pour le pipeline
- La désignation directe s'adapte difficilement au concept de client/serveur

Exemple : clients/serveur

- Un client envoie des demandes de service
- Le serveur reçoit de façon répétitive les demandes, il les exécute et il retourne une réponse.
- Le serveur reçoit les messages dans l'ordre d'arrivée sans se préoccuper de sa provenance

Exemple : clients/serveur

- La désignation directe est incapable d'implanter cette relation avec la réception sélective
- Solution : ?

Exemple : clients/serveur

- La désignation directe est incapable d'implanter cette relation avec la réception sélective
- Solution : ?
  - introduction d'une primitive de réception non-sélective

    `receive` *message*

Désignation directe

- Avantages : simple
- Inconvénients
  - changement de processus
  - serveurs multiples

Boîte aux lettres

- Une boîte aux lettres peut apparaître dans le `send` et dans le `receive`

  `send` *`message`* `to` *`boîte`*

  `receive` *`message`* `from` *`boîte`*

- Un message peut donc être reçu par tous les processus ayant accès à la boîte
- Convient mieux à la relation client/serveur

Boîte aux lettres (port global)

- Propriétés du canal
  - La communication implique l'existence d'une boîte aux lettres
  - Un canal peut s'associer à plus de deux processus
  - Plusieurs canaux peuvent exister entre une même paire de processus
  - Le canal peut être unidirectionnel ou bidirectionnel

Boîte aux lettres (port global)

- Synchronisation requise pour les accès simultanés
- Opérations requises pour manipulation

Boîte aux lettres : manipulation

- Propriété du système ou du processus ?
  - système $\rightarrow$ appels systèmes
  - processus $\rightarrow$ support du langage

Boîte aux lettres du processus

Propriété du processus !

- On distingue propriétaire et utilisateur
  - le propriétaire est l'unique récepteur
  - les utilisateurs envoient des messages
- Chaque boîte possède un seul propriétaire
- Lorsque le processus disparaît, la boîte disparaît
- Désignation du propriétaire : déclaration de variables, ...

Exemple : Boîtes aux lettres en SR

```
resource cs2 ()
  op add(id:int; val1:int; val2:int), resultats [3](rep:int)

  process client ( i := 1 to 3)
     var reponse : int
     write ( " Client " , i , " envoie sa demande " )
     send additionne (i ,i+2 ,i+3)
     receive resultats [ i ]( reponse )
     write ( " Réponse client " ,i ," = >" ,reponse )
  end

  process server
     var id : int , val1 : int , val2 : int , rep : int
     do true ->
         receive additionne ( id, val1, val2 )
         rep := val1 + val2
         send resultats [ id ]( rep )
     od
  end
end
```

Boîte aux lettres du système

Propriété du système !

- Le système fournit des appels suivants : création, destruction, envoi, réception.
- Le créateur est le propriétaire par défaut
- La notion de propriété est transférable et partageable
- La destruction de la boîte est explicite (si oubli $\rightarrow$? ? ?)

Boîte aux lettres du système

- Les boîtes aux lettres sont coûteuses à implanter
- Que faire lors d'un envoi sur une boîte distribuée ?
- Que faire lors de la réception sur un boîte distribuée ?

Port

- Cas particulier des boîtes aux lettres
- Un seul processus peut recevoir d'un port
- Avantages ?
- Inconvénients ?

Problèmes des ports

- Deux façons de les utiliser :
  - orientation vers les messages
  - orientation vers les liens
- Désignation (statique ou dynamique)

Degrés de synchronisation

- La transmission du message assure automatiquement la synchronisation
- Il existe différents degrés de synchronisation souvent reliés à la capacité du système d'emmagasiner ou non les messages

Synchronisation (Envoi)

- Processus expéditeur bloque jusqu'à réception (aucun emmagasinage, rendez-vous)
- Processus expéditeur ne bloque jamais (espace infini)
- Processus expéditeur bloque selon la régulation du flux des messages
  - nombre maximum de messages par port, par processus ou dans le système.
- Processus expéditeur bloque jusqu'à la réception d'une réponse

Synchronisation (Réception)

- Récepteur bloque jusqu'à l'arrivée du message
- Récepteur ne bloque pas :
  - on doit prévoir une primitive pour attendre la réception

Nature des messages

- Les messages peuvent être typés
- La taille peut être fixe ou variable
- Il peut exister plusieurs formats de messages

Protection

- Dans un système orienté vers le passage de messages on doit se protéger des influences indirectes :
  - messages erronés
  - fin de processus
  - erreurs de communication
  - ...

Exemples

- Si un des processus interlocuteurs termine de façon impromptu.
  - attente d'un message d'un processus qui n'existe plus.
  - envoi d'un message à un processus qui n'existe plus
- Filtrage des messages (protection)
- Erreurs de communication
  - corruption d'un message
  - perte de messages
  - violation du protocole
  - duplication de messages

Détection des erreurs

- Deux questions : Qui détecte les erreurs et comment ?
- Qui ? ? ?

Détection des erreurs

- Deux questions : Qui détecte les erreurs et comment ?
- Qui ? ? ?
  - le système détecte et effectue la reprise
  - le processus détecte et effectue la reprise
  - le système détecte et le processus effectue la reprise

Détection des erreurs

- Deux questions : Qui détecte les erreurs et comment ?
- Qui ? ? ?
  - le système détecte et effectue la reprise
  - le processus détecte et effectue la reprise
  - le système détecte et le processus effectue la reprise
- Comment ? ? ?

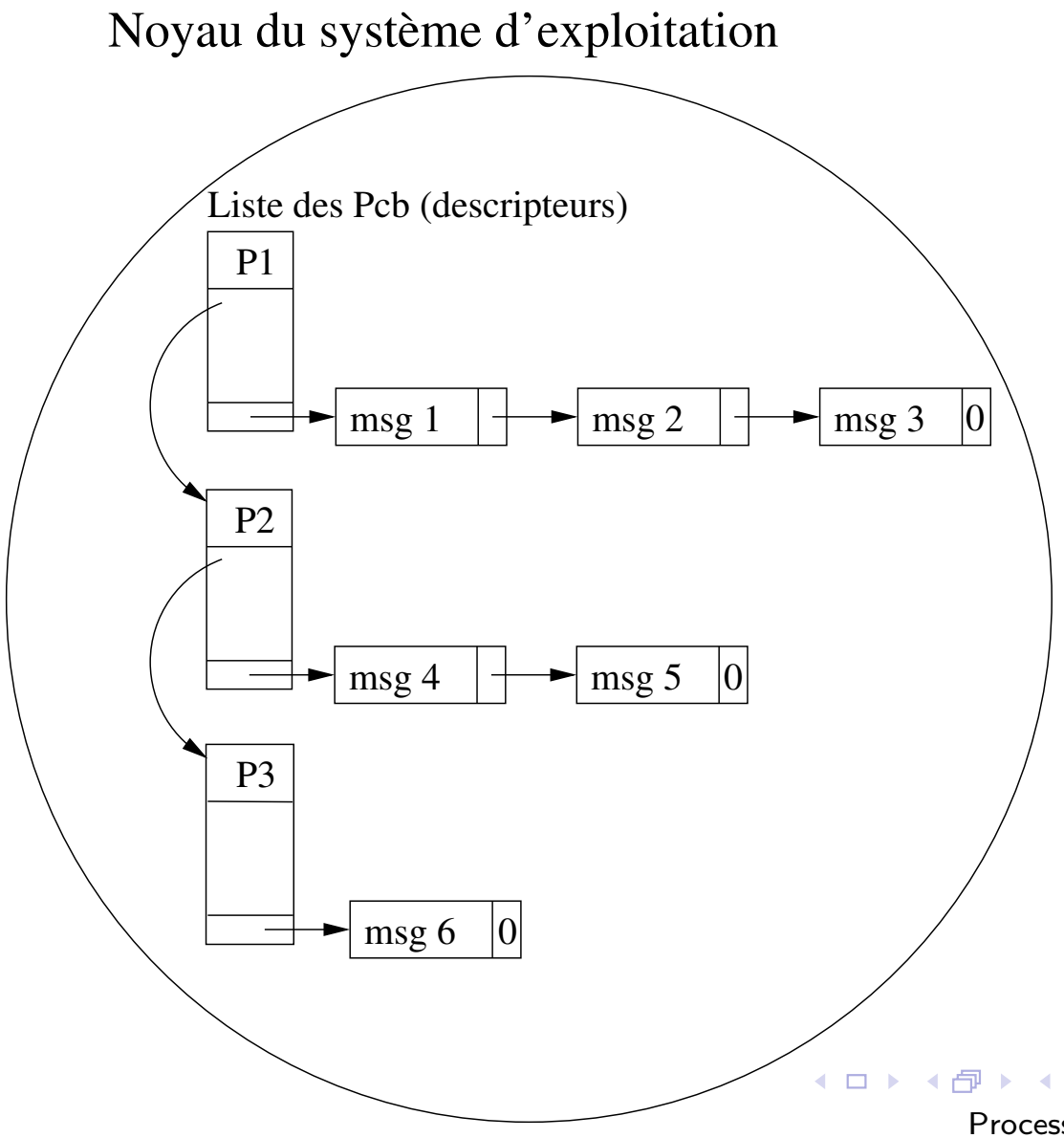
Détection des erreurs

- Deux questions : Qui détecte les erreurs et comment ?
- Qui ? ? ?
  - le système détecte et effectue la reprise
  - le processus détecte et effectue la reprise
  - le système détecte et le processus effectue la reprise
- Comment ? ? ?
  - horloge de garde
  - ramasse miette
  - code détecteur et correcteur

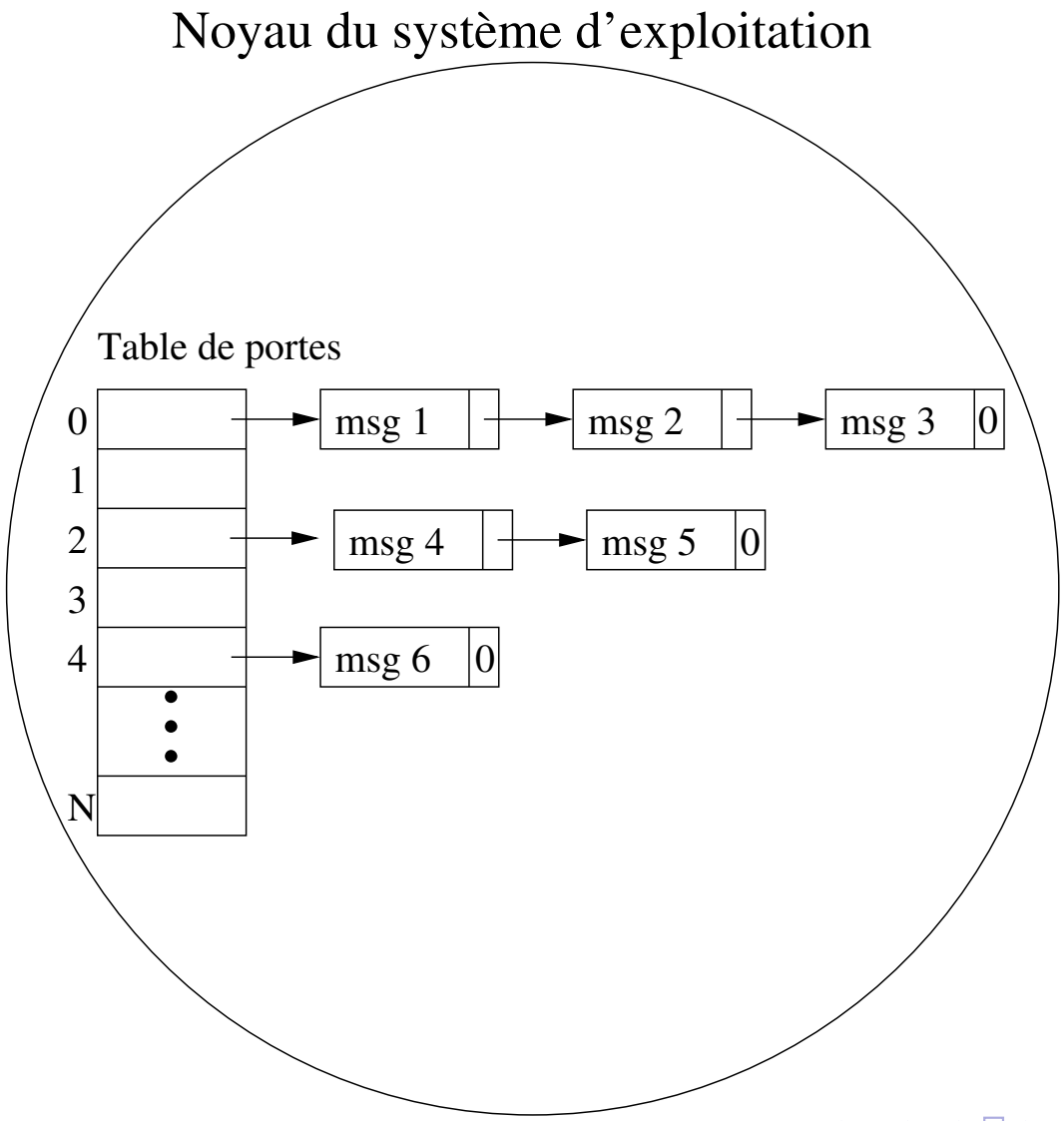
Implantation

- Généralement implanté par le noyau
- Utilise le PCB (bloc de contrôle)
- Deux cas à traiter :
  - si emmagasine les messages, alors une file par PCB synchronisé par des sémaphores
  - si rendez-vous, le système tient une table de rendez-vous

Implantation



Implantation



Implantation

Émetteur la commande	Cible de la commande	Commande
P1	P2	`send`
P3	P4	`receive`
P5	P6	`send`
P7	P8	`send`
P9	P10	`receive`

Implantation

- Transmission des messages
  - par contenu (on recopie tout le message)
  - par adresse

Chapitre 5 - Communication inter-processus

Futures et promesses

- Ce sont des abstractions utilisées pour la synchronisation et la communication
- Elles permettent d'écrire des fonctions asynchrones
- Elles ressemblent à des ports de communication

Définition

- Future $\equiv$ valeur qui sera éventuellement disponible.
- Fonctionnement :
  - «(future X)» retourne immédiatement une «future» pour la valeur de l'expression X
  - L'évaluation de X est lancée en parallèle (fil ou événement)
  - Lorsque l'évaluation de X termine la valeur remplace la future.

Définition

- Une future est une variable à assignation unique
- Elle a au moins deux états : «complété» ou «indéterminé»
- Une lecture d'une future à l'état indéterminé est bloquante

Exemple en Scala

```
val f = Future {
Http("http://.....").assString
}

f onComplete {
    case Succes(response) => println(response.body)
    case Failure(t) => println(t)
}
```

Exemple

```
#include <thread>
#include <iostream>
#include <future>

void fonction1()
{
    std::cout << "Fil d'execution..." << std::endl;
}

int main()
{
    auto future1 = async(launch::async, fonction1);

    future1.get();
    cout << "Le futur est arrive ..." << endl;
    return 0;
}
```

Exemple

```
#include <thread>
#include <iostream>
#include <future>

using namespace std;

int main(){
    auto future1 = async(launch::async,
                         [](){cout << "Fil 1" << endl;});

    future1.get();
    cout << "Le futur est arrive ..." << endl;
    return 0;
}
```

Exemple

```
#include <thread>
#include <iostream>
#include <future>
#include <chrono>
#include <vector>
using namespace std;
int main()
{
 vector<future<int>> mes_futures;

 for (int i = 0; i < 10; ++i)
 {
   mes_futures.emplace_back(async(launch::async,[](int par)
   {
       this_thread::sleep_for(chrono::seconds(par));
       return par;
   }, i));
 }
```

Exemple

```
    cout << "Debut test de fin" << endl;

    for (auto &future : mes_futurs)
    {
        cout << future.get() << endl;
    }

    return 0;
}
```

Exemple

```
#include ...
using namespace std;

int main(){
    auto promesse = promise<std::string>();
    auto producteur = thread([&] {
            promesse.set_value("Bonjour.....\n ");
        });
    auto futur1 = promesse.get_future();
    auto consommateur = thread([&] {
            std::cout << futur1.get();
        });

    producteur.join();
    consommateur.join();
    return 0;
}
```

Chapitre 5 - Communication inter-processus

En pratique ...

- Systèmes d'exploitation
  - communication locale
  - communication réseau
- Middelware
  - MPI
  - MOM (IBM Websphere MQ, Microsoft MSMQ, Apache ActiveMQ et Kafka, Oracle OpenJMS, ...)
  - ...

Chapitre 5 - Communication inter-processus

Conclusion